Karlsruhe Institute of Technology

Modeling Variations in Load Intensity Profiles

Master Thesis of

Jóakim Gunnarsson v. Kistowski

At the Department of Informatics
Institute for Program Structures
and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Advisor:	Dipl.-Inform. Nikolas R. Herbst
Second advisor:	Dipl.-Inform. Rouven Krebs, SAP

Duration: October 31st, 2013 – March 20th, 2014

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, March 20th, 2014

. . . .

(**Jóakim Gunnarsson v. Kistowski**)

Zusammenfassung

Heutige Software Systeme müssen zuverlässige Leistung unter stark variierenden Lastintensitäten liefern. Gleichzeitig sollen sie dynamisch allozierte Ressourcen effizient ausnutzen. Konventionelle Benchmarking Umgebungen unterstützen die Emulation von solch dynamischen Lastprofilen nur teilweise. Industrielle Benchmarks nutzen typischerweise Arbeitslasten mit konstanter oder schrittweise ansteigender Lastintensität. Alternativ wiederholen sie einfach vergangene aufgenommene Lastverteilungen.

Als Ergebnis dieser Beobachtungen stellt diese Arbeit fest, dass Mittel zur Definition von flexiblen Lastprofilen erstellt werden sollten. Dieser Bedarf wird durch die Einführung von Metamodellen auf zwei verschiedenen Abstraktionsebenen adressiert. Auf der niedrigeren Abstraktionsebene bietet das *Descartes Load Intensity Meta-Model* (DLIM) eine strukturierte und zugängliche Herangehensweise, um Lastintensitätsprofile durch Editieren und Kombinieren von mathematischen Funktionen zu beschreiben. Das *high-level Descartes Load Intensity Meta-Model* (hl-DLIM) ermöglicht die Beschreibung realitätsnaher Lastintensitätsprofile mit der Hilfe von einigen wenigen Parametern, welche saisonale Muster, Trends, Bursts und Rauschen beschreiben. Mit Hilfe dieser Parameter kann es die am häufigsten vorkommende Teilmenge der insgesamt möglichen Lastintensitätsprofile beschreiben.

Für den Einsatz der Meta-Modelle, wurde im Rahmen dieser Masterarbeit LIMBO entwickelt - eine Eclipse-basierende Umgebung für die Modellierung von variablen Lastprofilen, basierend auf DLIM und hl-DLIM als zugrundeliegende Modellierungsformalismen. LIMBO bietet eine Visualisierung für DLIM Instanzen, sowie einen Modellerstellungs-Wizard auf Basis der hl-DLIM Parameter. Weiterhin bietet es drei automatisierte Modell-Extraktions-Prozesse an, mit denen DLIM und hl-DLIM Instanzen von existierenden Messungen von Ankunftsraten erstellt werden können. Es bietet auch eine Modell-zu-Modell Transformation von hl-DLIM zu DLIM an.

Durch den Vergleich von neun unterschiedlichen realen Lastaufzeichnungen, welche über Zeiträume von zwei Wochen bis sieben Monaten gemessen worden sind, mit den Modell-Instanzen, die aus ihnen extrahiert worden sind, zeigt diese Arbeit, dass beide Metamodelle in der Lage sind Lastintensitätsprofile, wie sie in der realen Welt auftreten, mit einer akzeptablen Genauigkeit bei einem durchschnittlichen Median-Fehler von 19.9% zu erfassen. Zusätzlich evaluiere ich die Nutzbarkeit und Zugänglichkeit von LIMBO anhand eines Aufgaben- und Fragebogens, welcher von acht Informatikern aus fünf verschiedenen Organisationen oder Unternehmen aus dem Performance-Engineering-Umfeld bearbeitet und beantwortet wurde.

Abstract

Today's software systems are expected to deliver reliable performance under highly variable load intensities while at the same time making efficient use of dynamically allocated resources. Conventional benchmarking frameworks provide limited support for emulating such highly variable and dynamic load profiles and workload scenarios. Industrial benchmarks typically use workloads with constant or stepwise increasing load intensity, or they simply replay recorded workload traces.

Based on this observation, I identify the need for means allowing flexible definition of load profiles and address this by introducing two meta-models at different abstraction levels. At the lower abstraction level, the *Descartes Load Intensity Meta-Model* (DLIM) offers a structured and accessible way of describing the load intensity over time by editing and combining mathematical functions. The *high-level Descartes Load Intensity Meta-Model* (hl-DLIM) allows the description of load variations using few defined parameters that characterize the seasonal patterns, trends, bursts, and noise parts. Using these parameters, hl-DLIM is capable of describing a subset of most common load intensity variations.

During the work on this thesis I developed LIMBO - an Eclipse-based tool for modeling variable load intensity profiles based on DLIM and hl-DLIM as underlying modeling formalisms. LIMBO provides visualization for DLIM instances and a model creation wizard based on hl-DLIM parameters. It also offers three automated model extraction processes with which to extract DLIM and hl-DLIM instances from existing arrival rate traces. It also offers a model-to-model transformation from hl-DLIM to DLIM.

I demonstrate that both meta-models are capable of capturing real-world load profiles with acceptable accuracy, having an average median deviation of 19.9% from the original trace. This is done by comparing nine different real life traces, which were measured over duration of two weeks to seven months, to their respective model instances as extracted by the automated model extraction processes. I also evaluate the usability and accessibility of LIMBO based on a questionnaire which was answered by eight computer scientists from five different organizations within the performance engineering community.

Publications and Talks

1. Refereed Conference / Workshop Papers

1.1. Published

- [vKHK14c]

  J. G. von Kistowski, N. R. Herbst, and S. Kounev, "LIMBO: A Tool For Modeling Variable Load Intensities (Demonstration Paper)", in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014).* ACM, March 2014.

- [vKHK14b]

  J. G. von Kistowski, N. R. Herbst, and S. Kounev, "Modeling Variations in Load Intensity over Time", in *Proceedings of the 3rd International Workshop on Large-Scale Testing (LT 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014).* ACM, March 2014.

1.2. Under Review

- [vKHK14a]

  J. G. von Kistowski, N. R. Herbst, and S. Kounev, "Automatic Extraction of Load Intensity Profiles using the Descartes Load Intensity Meta-Model", in *Proceedings of the 11th International Conference on Autonomic Computing (ICAC 2014).* USENIX, June 2014, submitted on March 5th, 2014.

2. Invited Talks

- "LIMBO - A Load Intensity Modeling Platform", at the *SPEC RG Annual Meeting 2014*, Dublin. March 26th, 2014.

Contents

1. Introduction

Today's cloud and web-based IT services need to handle huge numbers of concurrent users. Customers access services independently of one another and expect reliable quality-of-service under highly variable and dynamic load intensities. In this context, any knowledge about a service's load intensity profile is becoming a crucial information for managing the underlying IT resource landscape.

Users use cloud and web-based IT services independently of one another and cause high variability in the load intensity of the service. Considering the high amount of independent users, one would assume that these load intensity variations are entirely random. This is not the case however, since user behavior is influenced by human habits, trends, and events. Factors such as time of day, time of the week, and/or current events can and do play a role when it comes to the number of requests the cloud service has to handle.

Benchmarking uses work units that are deployed on a test system in order to measure the system's performance properties. In many cases, such as web or cloud systems, multiple work units are deployed concurrently or sequentially to stress the system under test (SUT) in a realistic manner. The rate at which a defined class of work units arrives at the SUT is called **arrival rate**.

Cloud system benchmarks do not factor in this high load intensity variability. Common benchmarking frameworks such as Faban[1], Rain [BLY^{+}10], and JMeter [Hal08] allow job injection rates to be configured either to constant values, stepwise increasing rates (e.g. for stress tests), or rates based on recorded workload traces. This work aims at closing the gap between highly dynamic real-world load intensity profiles and the currently insufficient flexibility in handling variable load profiles in the benchmarking and dynamic system management domains.

To close the gap between the highly dynamic real-world load intensity variations and the currently insufficiently variable load intensities used for benchmarking, I create load intensity models for two major use-cases: Firstly, I support the creation of custom load intensity behaviors, specifically designed to illicit certain system behavior, such as dynamic resource allocation on elastic systems. Secondly, I realize a (partly) automatic approximation of existing load intensity traces in form of a model instance. I propose two meta-models at different abstraction level: The high-level Descartes Load Intensity Model (hl-DLIM) allows the description of load variations using few defined parameters that characterize the

[1]Faban `http://faban.org`

seasonal patterns, trends as well as bursts and noise parts. At the lower abstraction level, the Descartes Load Intensity Model (DLIM) offers a structured and accessible way of describing load intensity profiles over time by editing and combining mathematical functions. I also provide a transformation from hl-DLIM to DLIM model instances for the support of further model refinements and automatic DLIM calibration features to be applied at run-time as envisioned for future work.

For the handling and instantiation of the proposed load intensity models, I introduce the LIMBO toolkit[2] - an Eclipse-based tool for handling and instantiating load intensity models based on DLIM. LIMBO offers an accessible way of editing DLIM instances and extracting them from existing traces. It also supports using hl-DLIM parameters for easy creation of new DLIM instances through a model creation wizard.

LIMBO provides automated model extraction processes with which DLIM and hl-DLIM instances can be extracted from existing load variation traces. I present the *Simple Model Instance Extraction Process (S-MIEP)*, which is based on the approach taken in Breaks For Additive Season and Trend (BFAST) [VHNC10] and the *Periodic Model Instance Extraction Process (P-MIEP)* which features a wealth of repeating patterns, thus leading into the direction of load intensity forecasting. The third extraction process, the *high-level Model Instance Extraction Process (hl-MIEP)*, extracts hl-DLIM instances.

I evaluate the accuracy of DLIM and hl-DLIM as well as the extraction processes by comparing extracted model instances to real world traces. I measure absolute and relative arrival rate differences between trace and model instance, and I also apply a difference metric based on Dynamic Time Warping (DTW) [Mül07], which I created specifically for the comparison of load intensity variations. The evaluation shows that my optimal model extraction approach using S-MIEP has an average median accuracy of 19.9%, while at the same time performing 8354 as fast as the BFAST decomposition.

LIMBO usability is then evaluated based on a questionnaire, which was taken by members of the performance engineering community of different affiliations after working through a simple LIMBO tutorial.

I also perform a preliminary evaluation of the applicability of DLIM and the *Periodic Model Instance Extraction Process (P-MIEP)* for the purpose of load intensity forecasting. In this preliminary evaluation P-MIEP is able to forecast a month of requests with a median accuracy of 7.3%.

The remainder of this introduction is structured as follows: The following section outlines the motivation for the development of a meta-model for load intensity variations and the goals of the thesis. I describe some possible applications of a load intensity model and state the applications this thesis lays focus on. Sections 1.2 and 1.3 then describe the goals and benefits of the work in this thesis. Finally, Section 1.4 explains where the scope of this work ends.

1.1. Motivation

Some of the motivations for a load intensity model derive from the general motivations of model use in model-based and model-driven development.

Additional motivation for the Descartes Load Intensity Model (DLIM) can be found in the use cases, which I envision for DLIM and LIMBO.

[2]LIMBO `http://www.descartes-research.net/tools/`

1.1.1. Motivation for Meta-Modeling in General

Modeling of load intensity variations can help with the communication of load intensity behavior, as it enables people who are unaware of platform details to define and develop varying load intensities for benchmarking purposes. It also provides an abstraction level that can be used to describe and formalize already existing loads.

1.1.1.1. Model for Load Intensity Description

Load intensity description by modeling has several advantages. Firstly, it improves communication on the subject. A cloud platform customer might have a better ability of specifying the load distributions his platform is supposed to handle. Developers might then base tests on this more concrete description instead of having to rely on fuzzy statements and a few describing numbers.

1.1.1.2. Model for Load Intensity Generation

A model could also be used to describe the workload intensity of benchmarks, specifically. When this is the case, the model could be used to generate load intensities for benchmarking and load testing. A well defined and powerful model could enable the fast and easy creation of a variety of benchmark load intensities for a number of different scenarios, such as representative loads or specific problem oriented loads for a certain platform configuration.

1.1.1.3. Model for Load Intensity Pattern Formalization

Flexible composition of parametrized model elements allows to formalize characteristic patterns of load intensity behavior. I envision that the process of composition of model elements, with the purpose of creating a model based on an already existing load intensity (i.e. the decomposition of an already existing load intensity behavior into model elements) can lead to insight on different patterns and properties of the original load intensity behavior. This insight can then be formalized within the model elements' parameters as well as the composition of the elements themselves.

1.1.2. Use-Cases for a Load Intensity Model

A load intensity model can be applied in many contexts, some of which might not be directly apparent. The following sections describe a few use-case scenarios in which a load intensity model might be useful, in order to show the usefulness of load intensity modeling and to give the reader a few additional ideas on what to do with the Descartes Load Intensity Model.

1.1.2.1. Use-Cases in the Focus of this Work

This thesis places its focus on two major use-cases for the load intensity model:

- Creation of artificial load intensity variations for specific benchmarking purposes
- Extraction of existing load intensity variations from pre-existing traces.

Other possible use cases of the load intensity model will be part of future work.

1. **Creating artificial Load Intensities for Benchmarking**

   A model describing the load intensity variations over time can be used to create request or user arrival time-stamps that can then be used to define the beginning time of a unit of work within a benchmarking framework. This enables a user to use a multitude of different varying workloads, which in turn helps with the benchmarking of system properties that deal with such variations (such as elasticity).

This use-case describes the possibility of a custom created load intensity variation, that has been specifically designed to help with the benchmarking of such a property. Of course, this load intensity variation may be subject to additional requirements, such as representability.

2. **Creating a Load Intensity Model Instance from an existing Trace**

   A model instance can be used to describe a past real world load variation (within a certain error). Doing so can be useful in a number of sub-use-cases:

   - *Parametrization of Request / User Arrival Traces*

     Among others, Zakay et al. [ZF13] sample request traces for benchmark workload generation. When doing so several problems may arise. The trace might be taken from a system that is magnitudes larger than the test system on which the benchmark is to be executed. The trace might also have been taken over a long time period and has to be temporally compressed for the benchmark. When using a load intensity model instead of a simple trace, these problems become easily manageable. They can be managed either by modifying the model instance directly, or through parametrization of the request time-stamp generation (see Section 5.4.2).

   - *Anonymization of Request / User Arrival Traces*

     Request traces of real cloud or web based systems often include additional information that may contain information about the system's users. Even the exact time-stamps themselves may still provide a reader of the trace with the ability to extract information about the behavior of single users.

     An abstract load variation representation helps to minimize this problem. System providers, who are concerned about customer anonymity, might be more likely to provide usage information for research purposes in an abstract form as made possible by a load intensity model.

1.1.2.2. Additional Use-Cases

These additional use-cases are either derived from the two previous cases or constitute new approaches to the Load Intensity Model that may be useful to consider as part of future work.

1. **Load Intensity Forecasting**

   A model instance that has been derived from the incoming request trace of a currently running system might be used to predict future request intensity variations. This can then additionally help with the detection of unplanned events that deviate from the periodic model that most likely results from such an extrapolation. Such a forecasting mechanism could then be deployed on cloud systems. There it would help to improve dynamic resource management, by increasing the efficiency of elastic resource re-allocation.

2. **Anomaly Detection**

   The use-case of using a model for load intensity forecasting already hinted at the use of a load intensity model as a baseline of predicted system behavior. This baseline can also be used in other fields of computer science, since it can always be used for comparison against anomalies. In a security context, it might be used for finding access patterns that deviate from usual access patterns in the form of their load intensity as part of an intrusion detection benchmark as proposed in [MK12].

1.2. Goals

This thesis focuses on the use of the proposed meta-models for the creation of varying load intensity behavior. The goals are thus aligned with the properties that improve model instance creation. These properties are mainly:

- **Model applicability**: The model, as well as the provided modeling tools, are easy to use, even by untrained third party users. Users shall be able to create a model instance based on their specifications within relatively short time periods and without prior training.
- **Model accuracy**: The model is capable of accurately describing a desired load intensity behavior.

1.2.1. General Goals for Meta-Modeling

The meta-model captures a load intensity model, based on arrival rates with the goal of creating a benchmark based on an open workload. To this end, the meta-model is to be defined in a way that allows a benchmark based on the model to adequately fulfill the quality criteria as put forth by Huppler in [Hup09]:

- Relevancy
- Repeatability
- Fairness
- Verifiability
- Economy

While these properties are also highly dependent on the workload itself, the meta-model should aim to fulfill them for the benchmark's load intensity level.

The meta-model should create a deterministic model, ensuring repeatability. The only exception to the determinism are random noise model elements, which, while not being deterministic, should still be reproducible using the same seed and therefore enabling complete repeatability.

Benchmark relevancy, while being in the hands of the benchmark developer should be improved by creating a meta-model powerful enough to describe real world load intensities. The model should also be understandable enough to warrant its use in the first place, speeding up the benchmark development process and thus enhancing the economy requirement stated above.

1.2.2. Research Questions

During the course of this thesis, I answer the following research questions. They represent the major research goals to be achieved by each unit of work (Meta-Model Definition, Process Definition, Evaluation ...). In detail, the research questions are:

1. Is it possible to create a model to capture load intensity variations over time?
    a) Can this model be used for custom load intensity variation creation for specific benchmarking purposes?

   b) Can this model be used to model existing real world load intensity traces accurately?

2. Is it possible to formalize a process in order to extract model instances from existing load intensity traces?

   a) Can this process extract model instances accurately?

   b) Can this process extract model instances reliably?

   c) Which parts of the process require automation?

1.3. Benefits

The goals of this thesis provide benefits both towards creation as well as analysis of load intensity variation profiles.

1.3.1. Accurate Description of Load Intensity Variations

A model of load intensity variations can be used to describe the underlying variations. This description can be tailored towards human readability or modeling accuracy. In order to achieve both benefits, this thesis introduces two meta-models.

- DLIM provides an accurate way of describing load intensity variations, as shown in Section 7.1.
- hl-DLIM provides a less accurate, yet more understandable and intuitive way for load intensity description (see Section 7.2).

Description of these load intensity variations can lead to a better understanding of the involved patterns, as well as an easier identification of important or recurring components within these variations. They also enable easy sharing of load intensity variations.

1.3.2. Creation of specific Load Intensity Variations for Benchmarking

The DLIM and hl-DLIM meta-models as well as the LIMBO modeling platform enable the creation of specific load intensity variations. The major use-case for these variations is the use as input for benchmarking frameworks, as is supported by LIMBO. Benchmarks based on the created variations can then be used to test specific system properties, such as elastic resource allocation.

1.3.3. Automated Assistance for Load Intensity Analysis

LIMBO provides automated model instance extraction processes (see Chapter 6), which can be used to create DLIM and hl-DLIM instances from existing arrival rate traces. While this obviously enables the communicative aspect described in Section 1.3.1, it also enables further analysis of the load intensity variations through LIMBO's provided tools, such as the model decomposition visualization (see Section 5.7.3).

1.4. Limitations of Scope

The models in this thesis describe load intensities by arrival rates. They do not model user behavior, resource demand, response times, or any other workload related property. For this thesis, I assume that users in a typical cloud environment are completely unaware of one another and access a software service without having any knowledge of other users' behavior. Since the meta-models do not model the workload itself, they can only be used to define the intensity of an open workload. The open workload-approach is necessary, considering the execution time and behavior of each work unit remains unknown to the model.

Furthermore, DLIM and hl-DLIM do not announce the dispatched workload units. For the use of these models, I assume that work units induce statistically indistinguishable resource demands on the underlying hardware. Users of DLIM and hl-DLIM should thus assume a homogenous request / user mix when modeling load intensities or construct separate model instances accordingly.

2. Foundations

This chapter explains the central foundations needed for understanding the thesis. I discuss the differences between open and closed workloads and define what *load intensity* means for this work. I also describe the properties of load intensity that need to be considered for modeling.

2.1. Open and Closed Workloads

Schroeder et al. [SWHB06] define open and closed workloads as follows (also see Fig. 2.1):

- **Closed workload**: New job arrivals are only triggered by job completions.
- **Open workload**: New jobs arrive independently of job completions.

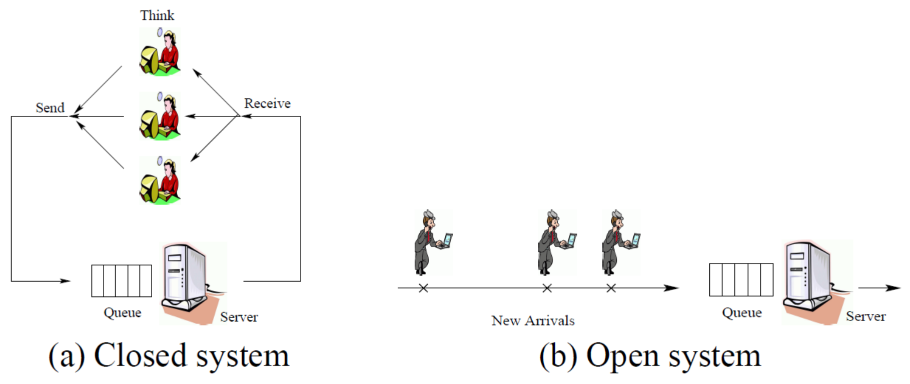


Figure 2.1.: Illustrations of the closed and open system models. Source:[SWHB06]

The models in this thesis do not make any assumptions about the jobs themselves. They do not know when they complete or if they complete at all. Considering that, they model the arrival of jobs completely independently of job completion. It is clear that the models of this thesis therefore describes the intensity of an open workload.

This makes sense, considering that users in a typical cloud environment are completely unaware of one another. They will access the service without having any knowledge of other users' behavior. In the interest of keeping the model as simple as possible, I have decided to omit the (in cloud computing scenarios) rare case of job completion dependent scheduling from the model all together.

Combining open and closed workload approaches is possible. In the context of this work that would entail a user or work unit model, which would be triggered by the load intensity model. The load intensity model would then trigger the arrival of closed workload batches, modeled using think-times, and life-times.

2.2. Load Intensity

In this thesis *load intensity* is the function describing *arrival rates* of workload units over time.

The *arrival rate* $r(t)$ at time t is defined as follows:

$$r(t) = R'(t)$$
$$\text{with } R(t) = |\{u_{t_0} | t_0 \leq t\}|$$

$R(t)$ being the amount of all *work units* u_{t_0}, with their respective *arrival time* t_0, that have arrived up until t.

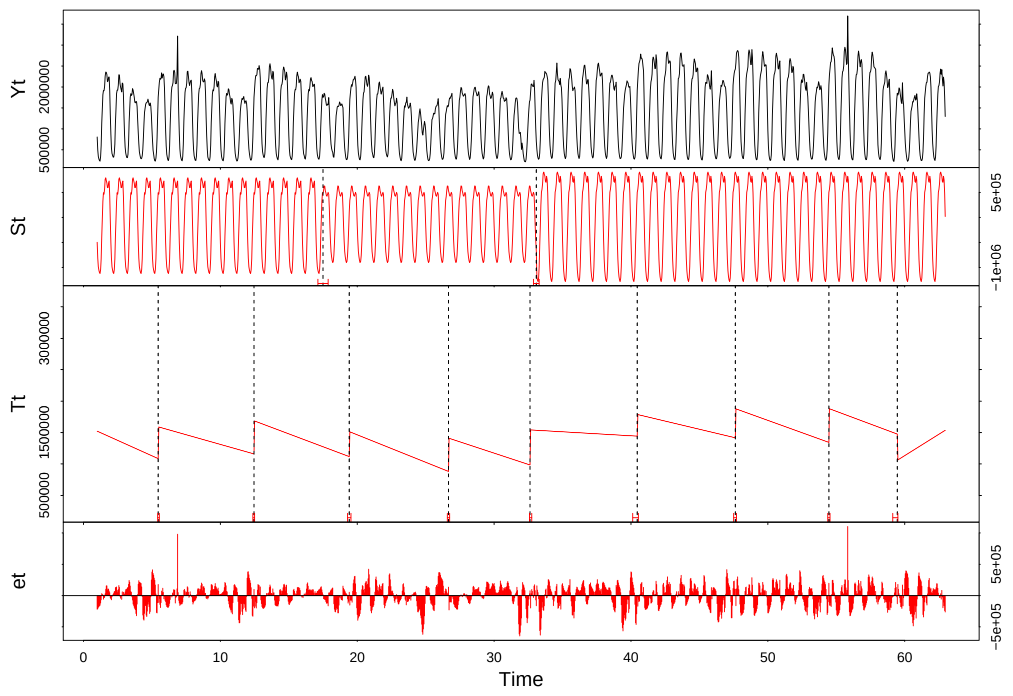


Figure 2.2.: The decomposition of a time series into seasonal, trend, and remainder, using BFAST [VHNC10].

The models of this thesis describe arrival rates of work units, with a major goal being the creation of a time series for workload execution. As such it is sensible to view the model as a specific way to describe a time series decomposition. For this reason, I chose to use the decompositional elements of time series as used by BFAST [VHNC10] as the foundation of the meta-model.

BFAST decomposes time series into the following functions (see Fig. 2.2):

- Seasonal: A possibly infinitely repeating function of the time series' seasonal characteristics (such as *sin* or *cos* functions).
- Trend: An additive trend that shows the change of the seasonal pattern over time.
- Remainder: The remainder, in the case of load intensity modeling this includes bursts and noise.

2.3. Pure Load Description vs. Workload Description

The Load Intensity Model resulting from this thesis' work is a model that describes load intensity only. It is independent of all considerations for the actual unit of work that is being executed. The load description used as part of the model thus only describes request arrival rates.

It does not model the following properties:

- Resource allocation and use
- Response, waiting, or service times
- The mix of different work units
- User behavior

Note that when modeling load intensity based on arrival rates using an open workload approach (as is done in this thesis), I do not model any user behavior including the time at which the session is ended by each user. Thus the model does not contain the number of concurrent users on the system. It models the number of concurrently arriving users.

Other models (see Chapter 3) model some or all of the excluded properties. They do however not go into detail with the description of varying load intensities.

A work that also takes varying load intensities into account is the thesis of Eike Schulz [Sch14]. He uses this thesis' approach as part of his workload model. His model consists of the following parts:

- **Application Model**
  - *Session Layer*: validity of service invocation sequences
  - *Protocol Layer*: service-related protocol details
- **Behavior Models**
  - *Markov chains* are used to capture probabilistic behavior of user types
  - *Distribution of think times*
- **Behavior Mix** is the relative frequencies of the mix of different Behavior Models, which make up the total workload
- **Workload Intensity**: (varying) number of virtual users as modeled by DLIM instances resulting from this thesis.

2.4. Self Similarity

Bai et al. [BS13] describe the property of self similarity for a process as follows: ”A process shows self-similarity implies the process is indistinguishable from its scaled versions obtained by averaging the original process within different observation time scales.”

This means that a load intensity model that is being stretched or compressed to another time scale has to scale its arrival rate behavior accordingly. If $s(t)$ is a linear function that scales the current time, and $ar(t)$ is the arrival rate function that results from the load intensity model, which is defined over the interval of $[0, t_{max})$, then the following must hold true for the scaled arrival rate functions $ar_s(t)$ (defined over the interval of $[0, s(t_{max}))$:

$$ar(t) = ar_s(s(t))$$

Forcing self similarity for the load intensity model is important for some use cases such as network traffic simulation. It also enables an intuitive understanding of the model's scaling behavior.

2.5. Meta-Model

A meta-model is a model describing another model. Thomas Stahl and Markus Völter [SV06] define a meta-model as describing the following parts:

Parts : Define the model's elements.

Rules : Define the rules of model validity.

Abstract Syntax : Describes the elements and their relations, representation independent.

Concrete Syntax : Describes the representation of model instances. In this case this is defined by the Eclipse Modeling Framework, which I use for meta-model definition (see the Tools Section 2.6).

Static Semantics : Contain the semantics independent of model execution (e.g. constraints).

Dynamic Semantics : Define the meaning of the model (in the case of this work: the load intensity variation, as described by the model).

Meta-models vary in the approach they take, they can attempt describe the problem-space itself and implicitly arrive at the solution or they can attempt to describe the solution and omit the problem. this seems to be a trade-off at times, with the two extremes being:

- A **problem oriented** approach: Creates a meta-model with the elements describing entities typically associated with load intensity behavior, such as bursts or sinusodial patterns.
- A **solution oriented** approach: Creates a less intuitive model based on freely configurable mathematical functions that can describe a wide range range of arrival rate distributions. This approach would promise a more powerful resulting model, which would be less intuitive to use.

2.6. Tools

The DLIM meta-model as well as the LIMBO modeling platform are based on the **Eclipse Modeling Framework (EMF)** [SBMP08]. EMF already provides a basic modeling environment, whereas additional tooling is added using the standard Eclipse plug-in development environment.

The Eclipse Rich Client Platform [MLA10] is also used to provide extension points for future work. These extension points can be used to add custom model exporters and model instance extractors to LIMBO. Default exporters and extractors are already implemented and shipped with LIMBO. These default exporters and extractors are capable of reading and writing both arrival rate and request time-stamp series.

The created time-series are in a .csv format. The goal of the exporters is to generate these time-stamps in a format, which is readable by benchmarking frameworks such as these:

- **Faban**[1] is a performance workload creation and execution framework for performance, scalability, and load testing of server applications.
- **Fincos**[2] is a benchmarking framework for evaluating complex event processing systems.
- **Rain** [BLY$^+$10] is a workload generation toolkit for cloud computing applications.
- **JMeter** [Hal08] is a Java-based framework widely used for load and stress testing purposes in general.

[1]Faban: `http://faban.org`
[2]Fincos: `https://code.google.com/p/fincos/`

3. Related Work

A number of approaches to generate workloads for cloud systems have been created already. These approaches differ from this thesis' approach in two key aspects, however. They are either statistical or they put key emphasis on the behavior of the actual units of work they dispatch (or both). As mentioned in Section 1.4, DLIM and hl-DLIM focus on the description of request or user arrivals, yet not their behavior after arrival. They do so in a deterministic fashion (excluding noise), which differs from the statistical modeling approaches.

Most of the related work can thus be grouped in at least one of the following three categories:

- **User behavior models**: [vHRH08], [RBG13], [BLY+10], [MAFM99].

  These models model work by modeling the behavior and tasks triggered by a user. This has a direct impact on the work type itself and differs greatly from my approach of only modeling user arrival rates.

  Models of this kind can however be easily combined with the Descartes Load Intensity Model, since they model what the user does after his/her arrival as modeled by the Descartes Load Intensity Model.

- **Workload modeling with a focus on work units and resource demands**: [CKKR12], [BC98].

  These models focus on the actual unit of work and model it in a way that allows for a specific resource use. These models differ from the approach taken in this work, since this thesis does not model the work unit at all. The use case of creating work for benchmarking of specific system behavior however, stays the same.

- **Statistical models**: [Fei02], [Li10], [MAR+03], [RLGPC+99], [BFF+10].

  These models try to extract workload intensity into a few statistical numbers that describe the workload. These approaches differ from my deterministic approach (although the DLIM *Noise* is not deterministic, but still reproducible), in that they do not capture the load intensity behavior changes over time.

3.1. User Behavior Models

Van Hoorn et al.[vHRH08] create an entire workload from his model. They model both the workload itself, as well as user behavior using Markov chains. This results in a *closed* *workload* model and thus differs strongly from the *open workload* approach employed in this work.

Roy et al.[RBG13] again model user behavior via Markov chains in the domain specific context of Video on Demand platforms.

Beich et al.[BLY$^+$10] create workloads specifically targeted for clouds. Their focus is on workload properties such as data usage hotspots with the goals of evaluating cloud specific properties such as elastic behavior and user separation. For this they take a *closed workload* approach at modeling user behavior. Their RAIN framework relates to the work in this thesis however in that their very basic user scheduler can take custom time-series as input.

Menascé et al. [MAFM99] analyze HTTP session logs in order to create Customer Behavior Model Graphs (CBMG), which they use to characterize user behavior within sessions.

3.2. Workload Modeling with Focus on Work Unit

Casale et al.[CKKR12] create bursty workloads by modeling the workload intensity. Their approach differs from this thesis' approach however, since it defines load intensity by arrival rates, whereas they define it by the number of concurrently running units of work times the resource usage of these work units. As a result their work focuses on assumptions on work unit resource usage and duration and the cross testing of these assumptions.

Barford et al. [BC98] model their workload with a focus on the type of requests to send. For this they evaluate workload properties, such as file popularity and distribution in order to create the typical challenges posted by Web workloads.

3.3. Statistical Models

Feitelson[Fei02] creates a statistical model for parallel job schedulers with the goal of system comparison through system performance evaluation. This differs considerably from this thesis' approach, especially considering the goals in mind. While DLIM and hl-DLIM should also be usable for system comparison, I envision that their deterministic natures enables easy communication and might also enable their use for intensity forecasting in the future.

Hui Li [Li10] models batch workloads for eScience grids. Batch workloads differ greatly from the cloud workload type that is the focus of this thesis. Li's focus is also more on statistical analysis than on deterministic workload creation. Li's work does however raise two points, which are fundamental to my work: *Inter-arrival times* and *Periodicity. Inter-arrival times* are the inverse to the arrival rates, which I am modeling and the observation that *Periodicity* exists is central to my Meta-Model.

Menascé et al. [MAR$^+$03] analyze workloads on multiple layers: Business, Session, Function, and the HTTP Request layer. The latter is related to my work. While analyzing HTTP Request volume, they notice several of the patterns, that where part of the motivations for this work (such as: lower request volumes on weekends). To this end they split the workload into two parts: noisy (non-predictable) oscillations and scales with strong correlations. On this they perform a statistical analysis in order to extract properties, such as variance and correlations.

Reyes at el. [RLGPC$^+$99] use a multi-layered approach towards www traffic modeling. Thy model *Session inter-arrival times* in their *Session Level*, which is the object of the Descartes Load Intensity Model as well. They do however model it as an exponentially distributed random variable, which differs greatly from my more deterministic approach, since they do not model the change of the inter-arrival rates over time.

Bodík et al. [BFF+10] model workload spikes with the goal of workload generation. They apply a single user model, in which one user sends a number of requests as defined by the model. This is effectively the open workload approach also taken by DLIM and hl-DLIM. They also focus on arrival rates and they evaluate their model using many of the workload traces used in this work (WorldCup98, Wikipedia project counts . . .). The major difference to my work however is both the focus on spikes, as DLIM and hl-DLIM model bursts as part of a greater model, thus putting less emphasis on them, as well as the statistical approach, which they base on request distributions within the spikes.

3.4. Others

Zakay et al. [ZF13] partition workloads into users and then sample workload traces for each user in order to describe user behavior. On a superficial level, this approach can also be taken using the Descartes Load Intensity Model by employing the *ArrivalRatesFromFile* model element for sampling. At closer inspection however, Zakay's work differs greatly from mine. Whereas I focus on creation and description of load intensities, Zakay et al. focus on the difficulties of sampling workload traces. Far beyond the naive approach of *ArrivalRatesFromFile* they employ advanced techniques, such as the partitioning of traces into subtraces for different users.

4. The Descartes Load Intensity Meta-Models

I describe load intensity using two models: The Descartes Load Intensity Model (DLIM) and the high-level Descartes Load Intensity Model (hl-DLIM). The base load intensity model provides structuring and compositing functionality for piece-wise mathematical functions, whereas the high-level model provides a set of parameters that describe load intensity variations in a more concise fashion on a more abstract level.

A Model-to-Model (M2M) transformation from the high-level Descartes Load Intensity Model to the Descartes Load Intensity Model is also provided.

4.1. The Descartes Load Intensity Meta-Model

4.1.1. General Considerations and Requirements

The Descartes Load Intensity Model (DLIM) describes request arrival rates over time. Specifically, the model is aimed at describing the abstract variations of work unit arrival rates into characteristic load intensity behavior. This can also be achieved by a piece-wise mathematical function. A mathematical function can be used to approximate any variable load intensities and describe them within a certain error.

DLIM offers a way to structure this underlying function for easy editing according to the following considerations and requirements and being centered around the following properties:

- *(Partial) Periodicity*: Model Elements may be repeating. Examples:
  - Peak usage during business hours for commercial systems.
  - Low usage during night times or weekends.
- *Flexible incorporation of unplanned events*: Obviously, the model has to account for non-periodic events, such as unplanned bursts, as well.
- *Compositability*: Elements may consist of a tree of other elements. The encompassing element may then modify its children. Examples:
  - A weekly pattern (more usage on weekdays, less on weekends) encompasses daily patterns (more usage at day, less at night).

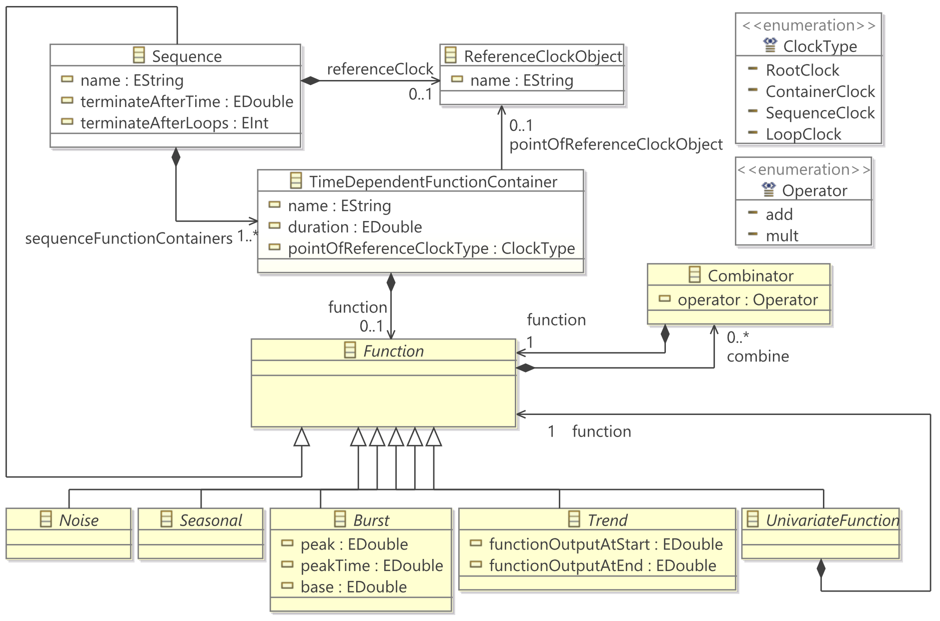


Figure 4.1.: The Descartes Load Intensity Meta-Model (DLIM) without the child implementations of the abstract *Noise*, *Burst*, *Seasonal*, and *Trend*.

– An unplanned burst adds onto the usual periodic usage.

Additional Requirements for the Meta-Model are:

- The functionality of each model element should be self-explanatory (as far as possible). As a result, all elements and attributes should have well defined and describing names.
- A user should be able to understand the model instance's behavior as easily as possible. Among other things, this means that there should be no hidden settings within model elements.

4.1.2. Sequence

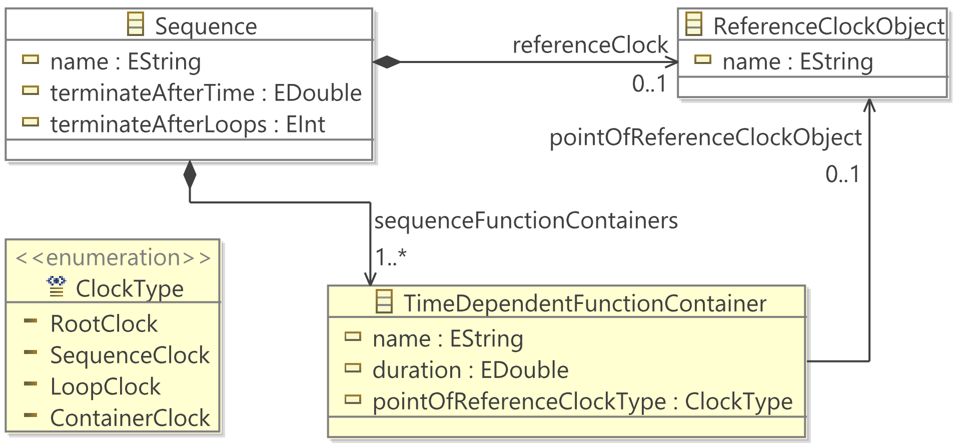


Figure 4.2.: The *Sequence* of the Descartes Load Intensity Model (DLIM).

The root element of any Descartes Load Intensity Model instance is a *Sequence* (see Fig. 4.2). The Sequence holds an ordered List of *TimeDependentFunctionContainers*, which describe the basic arrival rate functions (see Section 4.1.3). The *TimeDependentFunctionContainers* are executed in sequence. This execution repeats as many times as indicated by the *terminateAfterLoops* attribute. If the *terminateAfterLoops* attribute is unset (default value: -1), the *Sequence* repeats for the time indicated by the *terminateAfterTime* attribute. If both are set, the variation resulting in the shortest execution time is selected (meaning $finalDuration = min(terminateAfterLoops{*}loopDuration, terminateAfterTime)$.

Note that LIMBO does not allow infinite *Sequences* in order to guarantee benchmark termination.

Additionally, the *Sequence* holds a list of *Combinators* (see Section 4.1.4.2), which it inherits from *Function.*

4.1.3. TimeDependentFunctionContainer

The *TimeDependentFunctionContainer* contains functions that describe the arrival rate on a basic mathematical level, meaning that it contains one instance of a child of the abstract class *Function.* This can either be a *Sequence*, a child of *Noise*, *Seasonal*, *Burst*, or *Trend*, or a custom child of *Function*, such as *Polynomial* or *ArrivalRateFromFile.*

Any *Function* can be combined with other *Functions* using a *Combinator*, which results in a TimeDependentFunctionContainer carrying an entire Tree of functions.

The *TimeDependentFunctionContainer* describes its arrival rates for a set *duration*, after which the next *TimeDependentFunctionContainer* in the parent *Sequence's* list is executed.

TimeDependentFunctionContainers do not have an offset time to wait before execution. Every *TimeDependentFunctionContainer* is executed directly when the previous *TimeDependentFunctionContainer* ends. An offset can be achieved by creating a *TimeDependentFunctionContainer* without any defined functions. This *TimeDependentFunctionContainer* will then return the parent *Sequence's Combinator*'s neutral Element (1 for multiplication, 0 for addition and subtraction). The Reason for the absence of an offset is the goal of making model instances easier to grasp visually (Thus helping to fulfill the requirement of being able to grasp model behavior, see 4.1.1). By adding an additional *TimeDependentFunctionContainer*, optionally named by the DLIM user as offset, rather then editing an attribute, a reader can see the offset more easily. Thus, this decision is primarily based on the way in which attributes are presented in the EMF Model Editor.

The *ReferenceClockObject* and *ClockType* Attributes govern the input variable for all *Functions* within the *TimeDependentFunctionContainer*'s *Function* tree.

4.1.3.1. Reference Clocks

All functions DLIM are of the form $a = f(t)$, where a is the resulting arrival rate and t is the current time. The question, however, is: What specific time does t represent?

Reference Clocks, in the form of *ClockType* and *ReferenceClockObject* help to answer this question.

ClockType

ClockType is an enum with four possible values, which define the input time for the function:

- RootClock: The time since the beginning of the load intensity description (i.e. the time since the root *Sequence* started exection)

- ElementClock: The time since the current TimeDependentFunctionContainer started executing.
- SequenceClock: The time since the current *Sequence* (the *Sequence* containing the currently running TimeDependentFunctionContainer) started executing.
- LoopClock: The time since the current *Sequence's* first TimeDependentFunction-Container started its most current execution run.

ReferenceClockObject

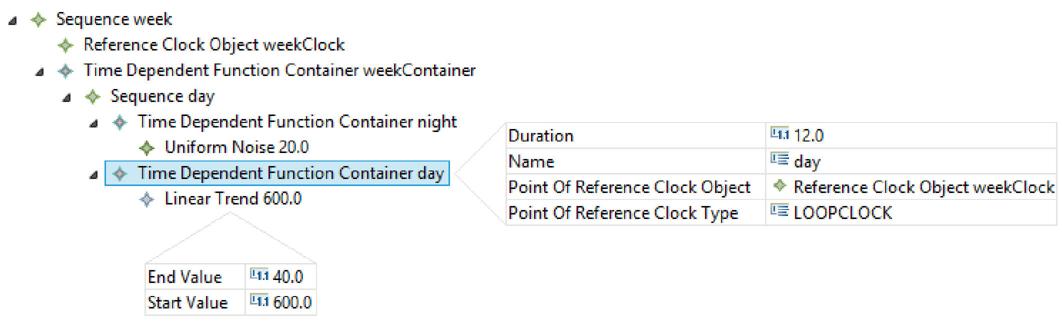
LoopClock and *SequenceClock* cause all *Functions* within the *TimeDependentFunctionContainer* to use a time defined by the *TimeDependentFunctionContainer*'s parent *Sequence* as input time. This parent *Sequence* may be the function of another TimeDependentFunctionContainer, however, which in turn is the child of another *Sequence*. This results in a tree of *Sequences* holding multiple *TimeDependentFunctionContainers*, which may again hold *Sequences* and so on.

When choosing the input time for a *Function*, a scenario might occur, in which it is necessary to use the time as defined by a *Sequence* that is not the direct parent of the *Function*'s containing *TimeDependentFunctionContainer*, but rather a node higher up in the tree that contains the *Function*, its parent *TimeDependentFunctionContainer* as well as its containing *Sequence*.

When using a time dependent on a parent *Sequence* as function input, it might be necessary to use the time dependent on a *Sequence* further up the tree, instead of the direct parent.

The *ReferenceClockObject* allows the user to do this. Every Sequence may contain an optional *ReferenceClockObject*, which a *TimeDependentFunctionContainer* may then reference. When using *SequenceClock* or *LoopClock* as function input, the *Sequence* chosen as the time reference is not the direct parent *Sequence* (as is the case, when no *ReferenceClockObject* is referenced), but the Sequence holding the *ReferenceClockObject* instead.

Note that the *ReferenceClockObject*'s parent Sequence must be currently executing when the function is being calculated. This means that the *ReferenceClockObject's* parent Sequence must be a node within the part of the tree that contains the current *TimeDependentFunctionContainer*.



- Sequence week
  - Reference Clock Object weekClock
  - Time Dependent Function Container weekContainer
    - Sequence day
      - Time Dependent Function Container night
        - Uniform Noise 20.0
      - Time Dependent Function Container day
        - Linear Trend 600.0

Duration	12.0
Name	day
Point Of Reference Clock Object	Reference Clock Object weekClock
Point Of Reference Clock Type	LOOPCLOCK

End Value	40.0
Start Value	600.0

Figure 4.3.: An Example model using a *Reference Clock Object*.

Figure 4.3 shows a simple artificial example of the usage of a *ReferenceClockObject*. This example features a simple day and night cycle over the course of a week. The night features random arrival rate noise, whereas the day features a linear decline from Monday morning to Sunday Evening. Since this decline depends on the time within the week rather than the time within the single day, the time of the *Sequence week* needs to be used as the input time for the *TimeDependentFunctionContainer day*, which contains the linear function. This is done by referencing the *Sequence week*'s *Reference Clock Object weekClock* in the *TimeDependentFunctionContainer* and setting the *Point Of Reference Clock Type* to *LOOPCLOCK*. This results with the time since the start of the current week as the input time for the linear declining function. The resulting arrival rates can be seen in Figure 4.4.

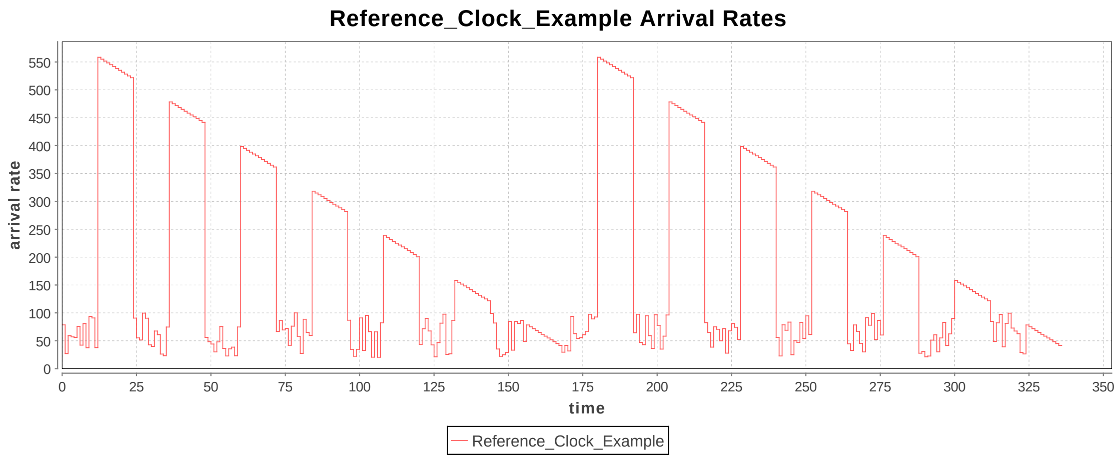


Figure 4.4.: The resulting arrival rates of the model in Figure 4.3, over the course of two weeks.

4.1.4. Function

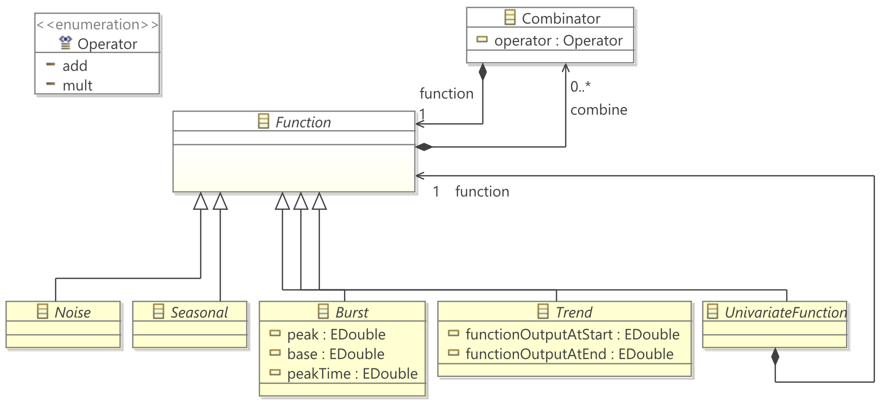


Figure 4.5.: The *Function* of the Descartes Load Intensity Model (DLIM).

The *Function* (see Fig. 4.5) is the abstract parent class to mathematical functions contained within the *TimeDependentFunctionContainer*.

4.1.4.1. Concrete Functions

The *Function* is abstract and cannot be instantiated. Instead a number of non-abstract children are provided that can be used as a *TimeDependentFunctionContainer*'s *Function* (For a complete list of currently available concrete *Functions* see Section 4.1.5). The most notable concrete *Function* is the *Sequence*, which effectively means that any *TimeDependentFunctionContainer* may hold a (complex) *Sequence* in its *Function* tree. This *Sequence* will then again contain further *TimeDependentFunctionContainers*, creating an even bigger tree of *Sequences* holding *TimeDependentFunctionContainers*, which hold *Sequences* and so on. Note that any *Sequence* contained within a *TimeDependentFunctionContainer* must be unique, thus preventing circular references.

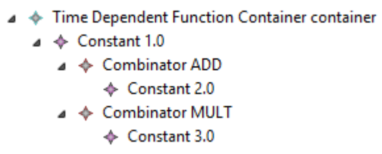
Most other concrete *Functions* fall into one of the following categories (each represented by their abstract super-class):

- *Seasonal*: Functions that describe seasonal, recurring patterns (such as *sin*).
- *Bursts*: Functions that describe singular bursts that reach a certain *peak* value at a *peakTime* and then return to the *base* value, from which they started.
- *Noise*: Functions that describe random Noise.
- *Trends*: Functions that describe monotonic trends. These Functions are intended to be applied to other *Functions* describing the underlying patterns (usually *Seasonal Functions*). They are interpolated *Functions*, described by their *start* and *end* values. They can also be used to approximate seasonal dummy functions by interpolating between the local minima and maxima within the seasonal function (e.g. see Section 4.2.5.1).

4.1.4.2. Combinator

The *Combinator* allows the combination of a *Function*'s arrival rates with the arrival rates generated by other concurrently running *Functions*. It contains the operator (+,-,*) with which to combine the arrival rates and a reference to a unique *Function* describing the arrival rates to be combined with.

It is also important to consider the order, in which the *Combinators* are applied to their parent *Function*. The *TimeDependentFunctionContainer* in Figure 4.6 helps to illustrate this problem.



```
Time Dependent Function Container container
  Constant 1.0
    Combinator ADD
      Constant 2.0
    Combinator MULT
      Constant 3.0
```

Figure 4.6.: Two different *Combinator*'s as children of a *Function*.

There are three approaches on the order in which the *Combinators* should be evaluated:

1. **Standard mathematical evaluation**:

   $$1_{parent} + 2 * 3 =_{EvaluateTree} 1 + (2 * 3) = 1 + 6 = 7$$

   The downside if this approach is obvious: The concept of the *Combinator* suggests the application with the parent *Function* containing the *Combinator*. In this case the *Constant 1.0 Function* holds both the *Constant 2.0*, as well as the *Constant 3.0 Functions*. The *Constant 3.0 Function* is not applied to its *Combinator's* parent *Function* however, but to its sibling *Constant 2.0* instead.

2. **In Order (Left to Right) Evaluation**:

   $$1_{parent} + 2 * 3 =_{EvaluateTree} ((1 + 2) * 3) = (3 * 3) = 9$$

   This approach solves the problem, that *Combinators* are not applied to their parent *Function*. However, the order of the *Combinators* still matters, which might lead confusing results for the modeler.

3. **Execute Multiplications First**:

   $$1_{parent} + 2 * 3 =_{EvaluateTree} 1 * 3 + 2 = 3 + 2 = 5$$

   This approach selects all multiplicative *Combinators* to be evaluated first. This results in an order independent way for all *Combinators* to be applied on their parent *Function*. LIMBO currently uses this approach.

As any *Function*, the *Function* contained within the *Combinator* may be a *Sequence*. In this case it is important to note that an expired *Sequence* (a *Sequence* that has passed beyond its termination-duration or maximum number of allowed loops) always returns an arrival rate of 0 or 1 depending on the operator of the *Combinator* holding it. This means that a *Sequence* that is a held by a *Combinator* with a multiplicative operator will return 1, whereas a *Sequence* held by a *Combinator* with an additive or subtractive operator will return 0. A *Sequence* may only be held by one *Combinator* in order to avoid conflicts as a result of this constraint.

4.1.4.3. Interpolation of Functions

Interpolated Functions are usually either *Trends* (with a *functionOutputAtStart* and *functionOutputAtEnd* value), or *Bursts* (with a *base* level, a *peak* value and a *peak time* at which this peak value is reached).

The *begin* and *end* time, at which the *start* and *end* values (or the *base* level) are defined, are denoted by the *Reference Clock* of the *TimeDependentFunctionContainer* holding this *Function*. The *begin* time is thus the time at which the reference clock starts running, whereas the *end* time is the time at which it stops (as defined by the duration of the parent *TimeDependentFunctionContainer* / the *Sequence* holding the referenced *ReferenceClockObject*).

4.1.5. Implemented Functions

The implemented Functions are a collection of parametrized functions that describe arrival rate variations. This list of functions is open and may be improved upon by adding additional functions as part of future work.

4.1.5.1. Seasonal

The following *Seasonal* functions have been implemented so far:

- ***Constant***: A constant function returning *value*.
- ***Sin***: A sin function defined by its min value, max value, period, and phase.
- ***Absolute Sin***: A sin function that returns the absolute value of the sin result.

4.1.5.2. Bursts

The following *Bursts* have been implemented so far:

- ***Linear Increase And Decline***: This function starts and ends at its *base* value at the respective times of 0 (as defined by the *Reference Clock*) and *duration* (as defined by the *Reference Clock*'s parent). It reaches its *peak* value at *peakTime* and interpolates linearly between these values.
- ***Exponential Increase And Decline***: This function is parametrized with the same attributes as its linear counterpart. The interpolation results in two exponential curves, however.

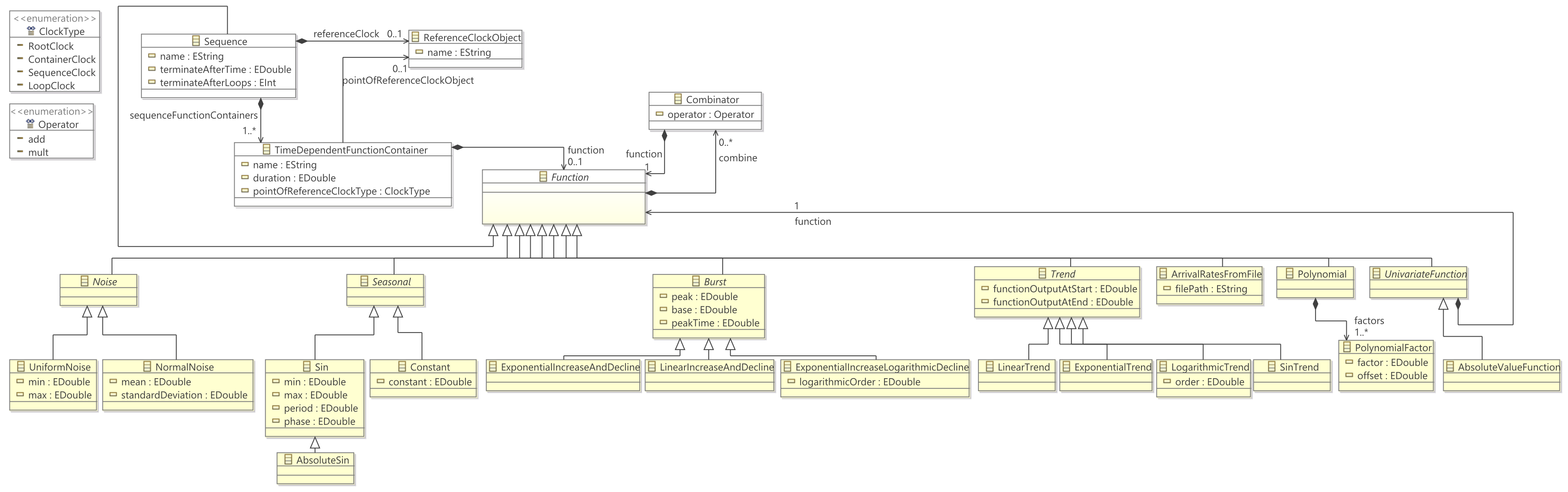


Figure 4.7.: The complete Descartes Load Intensity Meta-Model (DLIM), including all implemented functions at the time at which the work on this thesis is concluded (March 20th, 2014).

- ***Exponential Increase, Logarithmic Decline***: This function rises with an exponential curve, until reaching its *peak* value at *peakTime*, but declines using a logarithmic curve. The logarithmic interpolation uses the same approach as used by the logarithmic *Trend* in Section 4.1.5.4.

4.1.5.3. Noises

All Noises are generated based on distributions that use random variables provided by the *java.util.Random* random generator. This random generator is initialized using a fixed seed in order to guarantee reproducibility of its results. For different random distributions the distributions in the Apache Commons Math [Dev] library are employed.

The following *Noises* have been implemented so far:

- ***Uniform Noise***: This *Noise* generates arrival rates based on a uniform distribution between its *min* and *max* values.
- ***Normal Noise***: This *Noise* generates arrival rates based on a normal distribution, defined by its *mean* and *standard deviation*.

4.1.5.4. Trends

Trends always interpolate between a *functionOutputAtStart* at time 0 (as defined by the parent *TimeDependentFunctionContainers*'s *Reference Clock*) and a *functionOutputAtEnd* reached at time *duration* (as defined by the *Reference Clock*'s parent).

The following Trends have been implemented so far:

- ***Linear***: a linear interpolation between *functionOutputAtStart* and *functionOutputAtEnd*.
- ***Sin***: this Trend interpolates between the *functionOutputAtStart* and *functionOutputAtEnd* using the flank of a sine-curve.
- ***Exponential***: an exponential interpolation between *functionOutputAtStart* and *functionOutputAtEnd*.
- ***Logarithmic***: a logarithmic interpolation between *functionOutputAtStart* and *functionOutputAtEnd*. The shape of the curve is defined using the parameter *order*.

  Note that the logarithmic curve is always the same curve, starting at 1 and rising to *order*, which is then parametrized. This results in the following interpolation function being used:

  $$f(t) = start + (end - start) * \frac{1}{order} * log(\frac{t*(e^{order}-1)}{duration} + 1)$$

  This interpolation approach is used to prevent $e^{end-start}$ from taking too large a value, and thus resulting in a *double* overflow, as it would, when directly interpolating a logarithmic curve between the *functionOutputAtStart* and *functionOutputAtEnd* values. At the same time this solution limits the performance impact of this model element, thus helping to fulfill the performance and resource use requirement for the tooling environment (see Section 5.1).
  For reference, the original interpolation function, resulting in a *double* overflow for large values:

  $$f(t) = start + log(\frac{t}{duration} * (e^{end-start} - 1) + 1)$$

4.1.5.5. Polynomial

The *Polynomial* allows the definition of custom polynomials to serve as function. A *Polynomial* consists of a list of *PolynomialFactors* and calculates its end result as follows:

$$f(x) = \sum_{i=0}^{factors.size()-1} factor_i * (x + offset_i)^i$$

The offset is especially important, since it enables negative polynomial inputs and thus the creation of functions describing parables and other graphs requiring negative inputs. The offset of the 0th factor is never taken into account however, and thus always assumed to be 0. It is also not displayed in LIMBO's DLIM editor.

4.1.5.6. Arrival Rates from File

It is possible to specify a file containing arrival rates as input for a function. The file must follow the same format and pattern as arrival rate files generated by the time-stamp and arrival rate generator, which generates the time-stamp and arrival rate files from a Descartes Load Intensity Model.

The resulting function uses linear interpolation between the arrival rate values specified by the input arrival rate file. This enables the sampling of this function at points in time that were not specifically defined in the input arrival rate file. It should however be noted that the result of the linear interpolation might not always make sense depending on context (e.g. interpolation of Noise values is questionable). The user should always try to sample at the points in time that were specified in the original file.

LIMBO (see Chapter 5) includes a feature capable of generating these files from a series of time-stamps.

4.1.6. Validation Constraints

The DLIM EMF-Editor checks model multiplicity constraints automatically. Other validation constraints have been added manually:

- **Sequence::durationDefined()**: Checks if the *Sequence* duration has been properly defined. Returns **false** if *loops* is set to infinite (-1) and the duration is less or equal to 0.0.
- **TimeDependentFunctionContainer::durationGreaterZero()**: Returns **false** if the duration is less or equal to 0.0.
- **TimeDependentFunctionContainer::referenceClockInTreeNode()**: Returns **false** if the *Sequence* containing the referenced *Reference Clock Object* is not an (indirect) parent of the *TimeDependentFunctionContainer*.
- **Burst::peakTimeGreaterZero()**: Returns **false** if the *Burst*'s peakTime is less or equal to 0.

4.1.7. Technical Considerations

The *ecore* Meta-Model (see [SBMP08]) used by the generator and editor has a few additional elements that have not been described so far. The purpose of these elements is to enable functionality for the generator and editor without adding semantics to model. All of these elements are invisible to the user. They are mostly derived attributes.

4.1.7.1. Derived Time Attributes

Both *Sequence* and *TimeDependentFunctionContainer* hold a few derived Attributes that are used to store the points in time at which these are executed. These values are derived from the *duration* and *loops* attributes. They are:

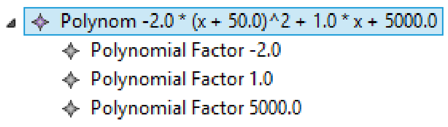
- **TimeDependentFunctionContainer::firstIterationStart**: The point in time at which the *TimeDependentFunctionContainer* first begins execution.
- **TimeDependentFunctionContainer::firstIterationEnd**: The point in time at which the *TimeDependentFunctionContainer* ends execution during its first execution run. This is usually also the *firstIterationStart* value of the next TimeDependentFunctionContainer.
- **Sequence::firstIterationStart**: The point in time at which the *Sequence* first begins execution.
- **Sequence::firstIterationEnd**: The point in time at which the *Sequence* ends its first execution run.
- **Sequence::loopDuration**: The duration of a single loop. This is the sum of the *durations* of the *Sequence*'s child *TimeDependentFunctionContainers*.
- **Sequence::finalDuration**: The actual duration for which the *Sequence* runs. This is calculated as $min(loops * loopDuration, duration)$.
- **ReferenceClockObject::loopTime**: The time since the *Sequence* containing the *ReferenceClockObject* started its last loop. This value is updated dynamically during time-series generation and accessed by a *TimeDependentFunctionContainer*, by setting its *PointOfReferenceClockType* to *LOOPCLOCK*.
- **ReferenceClockObject::seqTime**: The time since the *Sequence* containing the *ReferenceClockObject* first started executing $(currentTime - (Sequence :: start))$. This value is updated dynamically during time-series generation and accessed by a *TimeDependentFunctionContainer*, by setting its *PointOfReferenceClockType* to $SEQCLOCK$.

Note that while a *Sequence*'s actual duration can derive from its child *TimeDependentFunctionContainer*'s durations (by way of *loopDuration*), a *TimeDependentFunctionContainer*'s duration is always the value set by its attributes and is never derived from its (optional) child *Sequence*. This is done to minimize confusion and to allow the import of *Sequences* into *TimeDependentFunctionContainers* with shorter durations (In this case, the rest of the *Sequence* will not be executed. The *Sequence* will cut of at the end of the *TimeDependentFunctionContainer.*).

4.1.7.2. Edit Provider Labels

When being displayed in the EMF tree editor, the edit providers provide a label string for each element, with which it can identify itself and its contents to the user. For some elements this string is derived from their name (see *Sequence*), others have a single value, which gives conclusive insight into their functionality (such as *Constant*, which is defined by its one constant value) and is thus displayed in their label. Others however have multiple attributes, which all play an important part in the definition of the model element's behavior. In this case it is useful to derive the label from multiple attributes, in order to fill it with an overview of the element's attribute contents, so that they are visible at one glance. An example of this is shown in Figure 4.8.

The Edit Provider labels also add unit denominations to their respective attributes. *Sequence::terminateAfterTime* and *TimeDependentFunctionContainer::duration* both display **(s)** to show that they are defined in seconds.



Polynom -2.0 * (x + 50.0)^2 + 1.0 * x + 5000.0
- Polynomial Factor -2.0
- Polynomial Factor 1.0
- Polynomial Factor 5000.0

Figure 4.8.: The *label* representation of a polynomial function, derived from the function's *PolynomialFactors*.

4.1.7.3. ArrivalRatesFromFile

The *ArrivalRatesFromFile* reads a sequential arrival rate file and interpolates its values for random access. For this it needs to store the arrival rate values. It contains an ArrayList of arrival rate tuples, which is filled by calling its **ArrivalRatesFromFile::readFile()** method. The interpolated value at time x can then be retrieved by calling **ArrivalRatesFromFile::getArrivalRate(EDouble x)**.

4.2. The high-level Descartes Load Intensity Model

While the Descartes Load Intensity Model (DLIM) offers a convenient way of structuring and ordering functions for the description of load intensity variations, it offers little abstracted knowledge about those variations. Users may also find that the definition of mathematical functions using the Descartes Load Intensity Model requires too much work and time. The high-level Descartes Load Intensity Model (hl-DLIM) addresses this issue by describing load intensity variations with a limited number of parameters. These parameters can then be used to quickly define and characterize a load intensity model. A DLIM instance can be derived from an hl-DLIM instance using a Model-to Model-transformation.

Just as in BFAST [VHNC10], the high-level Descartes Load Intensity Model is split into a Seasonal Part and Trend Part. To those parts a burst and a noise component is added. As a result the model consists of the following components:

- **Seasonal**: As in BFAST, the *Seasonal* part denotes a repeating dummy function.
- **Trend**: The *Trend* is an overarching function containing several *Seasonal* functions, which is either multiplied or added to these seasonal functions (Other than in BFAST, where trends are always additive).
- **Burst**: An additive, recurring function that is added onto the *Seasonal* and *Trend* parts.
- **Noise**: Random uniformly distributed noise, which is added onto the function.

Other than base DLIM, the high-level Descartes Load Intensity Model (hl-DLIM) is not intended to cover all possible load intensity variations. It is primarily designed to allow for easy load intensity behavior creation and possibly a concise way of capturing the properties of the most common load intensity variations.

4.2.1. Seasonal Part

The *Seasonal Part* describes the underlying dummy function that repeats for every seasonal iteration (e.g. every day in a month long load intensity description). The high-level Descartes Load Intensity Model (hl-DLIM) describes it using the following parameters:

- **Period**: The duration of a single iteration of the seasonal function.

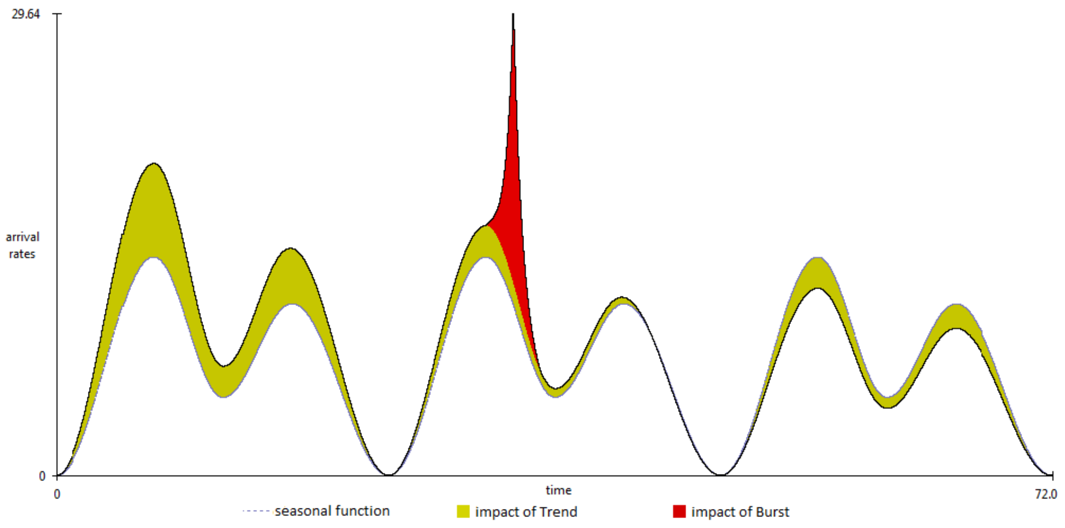


Figure 4.9.: The decomposition of a load intensity variation into its *Seasonal*, *Trend*, and *Burst* parts.

- **Number of Peaks**: The amount of arrival rate peaks within a single iteration of the seasonal function.
- **Base Arrival Rate Level**: The arrival rate at the beginning and end of a seasonal function iteration.
- **Base Arrival Rate Level between Peaks**: The arrival rate between two peaks.
- **First Peak Arrival Rate**: The arrival rate of the first peak.
- **Last Peak Arrival Rate**: The arrival rate of the last peak (if more than one peak exist).

  All other peak arrival rates are derived from *First Peak Arrival Rate* and *Last Peak Arrival Rate* using linear interpolation.
- **Interval containing Peaks**: The time interval during which the peaks are defined.

  This interval is centered around $Period/2$, meaning that the first peak is defined at $\frac{Period-Interval}{2}$, whereas that last peak is defined at $\frac{Period+Interval}{2}$.
- **Seasonal Shape**: The shape of the function interpolating between peaks and base levels. The Shape can be any *Trend Function*as defined in base DLIM.

4.2.2. Trend Part

The *Trend Part* describes the overarching function that describes the high-level Descartes Load Intensity Model. It can be either added or multiplied onto the *Seasonal Part*.

The *Trend Part* is defined using the following parameters:

- **Number of Seasonal Periods within one Trend**: The duration of each *Trend* segment. It must be a positive integer number. As a result, the *Trend* segments' duration is always a multiple of the *Seasonal Period*.
- **Trend Shape**: The mathematical function used to describe the *Trend* segments. The Shape can be any *Trend Function*as defined in base DLIM.

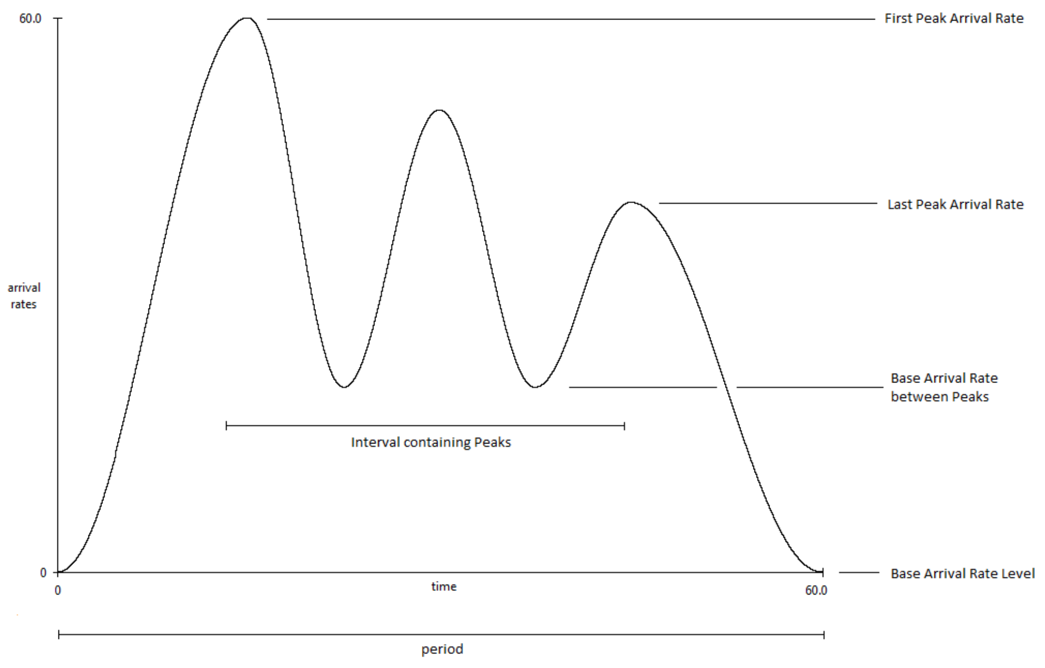


Figure 4.10.: The parameters defining the *Seasonal* part of the high-level Descartes Load Intensity Model (hl-DLIM).

- **Operator**: The operator (additive, multiplicative) with which the *Trend* is applied to the *Seasonal* part.
- **List of maximum target Seasonal Arrival Rate Peaks**: The arrival rate at the beginning and end of the *Trend* segments.

  The *Trend* segment functions are defined so that they always begin and end at the largest *Seasonal Peak*. As a result, the values contained in this list define the resulting maximum peak after *Trend* application at the corresponding point in time. The point in time at which each arrival rate in this list is defined is always the time of the largest peak in the $(ArrivalRateIndexInList * SeasonalPeriodsWithinOneTrend)$th *Seasonal* iteration.

4.2.3. Burst Part

Hl-DLIM allows the definition of recurring bursts. These bursts are added onto the existing arrival rate behavior. The *Burst Part* is thus defined using the following parameters:

- **First Burst Offset**: The time at which the first burst peaks.
- **Inter Burst Period**: The time between two peaks.
- **Burst Peak Arrival Rate**: The arrival rate added by the burst at its peak time.
- **Burst Width**: The time interval in which a burst is executed. The peak takes place at $BurstWidth/2$.

4.2.4. Noise Part

hl-DLIM allows the addition of a uniform distributed noise function. The function is defined by the following parameters:

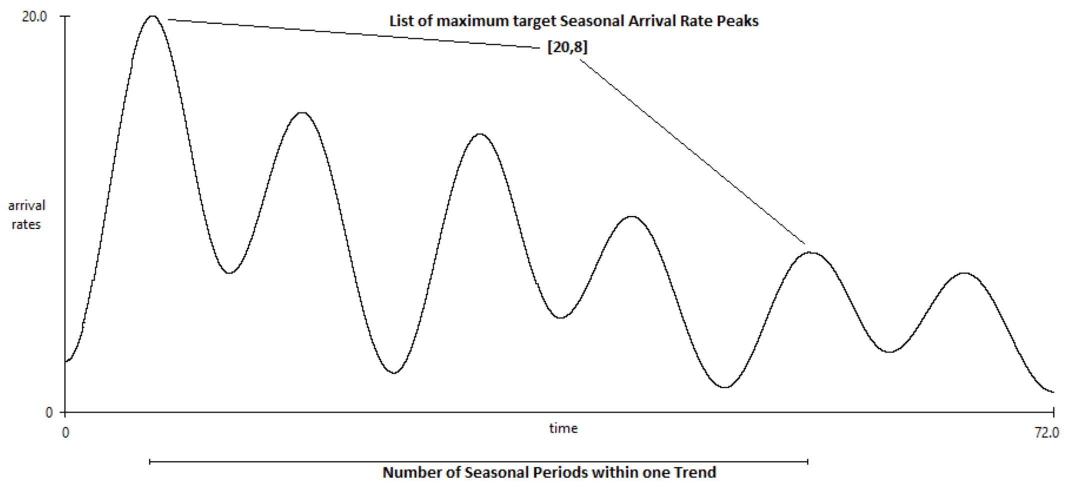


Figure 4.11.: The parameters defining the *Trend* part of the high-level Descartes Load Intensity Model (hl-DLIM).

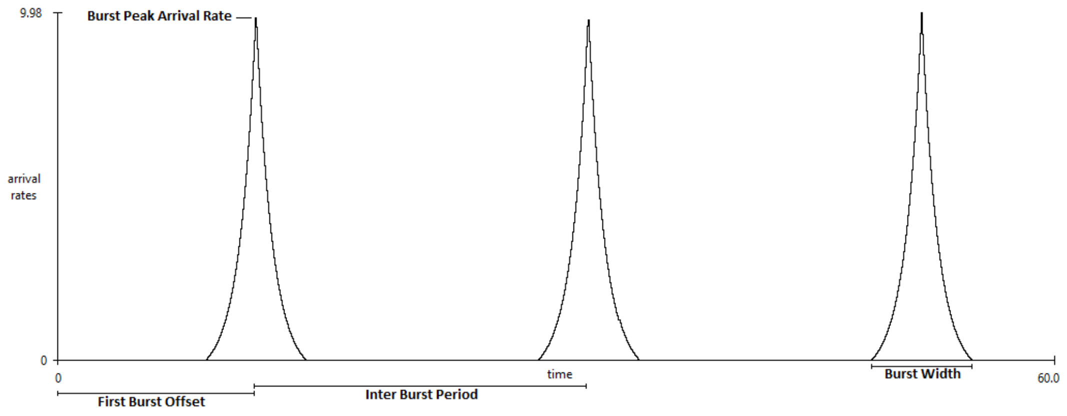


Figure 4.12.: The parameters defining the *Burst* part of the high-level Descartes Load Intensity Model (hl-DLIM).

- **Minimum Noise Arrival Rate**
- **Maximum Noise Arrival Rate**

Other noise distributions can easily be added in the Descartes Load Intensity Model, which results from the Model-to-Model transformation.

4.2.5. The hl-DLIM to DLIM Model-to-Model Transformation

Hl-DLIM instances are transformed into DLIM Instances using a Model-to-Model transformation.

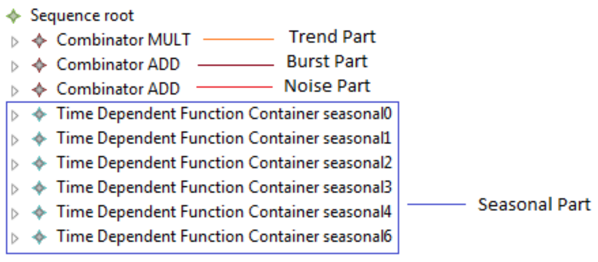

Sequence root
Combinator MULT
Trend Part
Combinator ADD
Burst Part
Combinator ADD
Noise Part
Time Dependent Function Container seasonal0
Time Dependent Function Container seasonal1
Time Dependent Function Container seasonal2
Time Dependent Function Container seasonal3
Time Dependent Function Container seasonal4
Time Dependent Function Container seasonal6
Seasonal Part

Figure 4.13.: A typical result of the Model-to-Model transformation from hl-DLIM to DLIM.

4.2.5.1. Transforming the Seasonal Part

The *Seasonal Part* is transformed into the *TimeDependentFunctionContainers* contained in DLIM's root *Sequence*. Each *TimeDependentFuctionContainer* then contains one *Function*, which interpolates between the curent *Base Level* and the next *Peak Arrival Rate* (or vice versa).

The concrete *Function* implementation used for this interpolation is defined using hl-DLIM's *Seasonal Shape* parameter.

The *TimeDependentFunctionContainers*' duration is derived from the point in time at which the peaks are defined.

4.2.5.2. Transforming the Trend Part

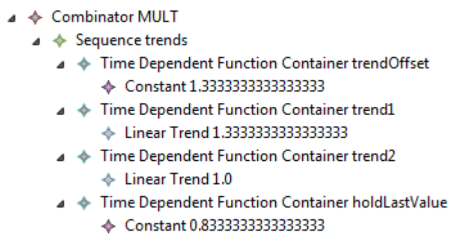
The *Trend Part* is transformed into a *Sequence*, which is contained by a *Combinator*. The *Combinator*'s *Operator* is defined by the *Trend Operator* parameter.

The *Trend* segment functions are contained by the *Sequence*'s *TimeDependentFunction-Containers*, which all have the duration of

$$SeasonalPeriod * NumberOfSeasonalPeriodsWithinOneTrend.$$

An offset *TimeDependentFunctionContainer*, containing a *Constant Function* is defined at both the beginning and the end of the *Sequence*'s *TimeDependentFunctionContainer* list. The duration of these offset containers guarantees that each *Trend* segment begins and ends at the point in time at which the largest *Seasonal* peak within the current *Seasonal* iteration is defined.

The duration of the entire *Trend* part is also set as the *terminateAfterTime* attribute of the root *Sequence*.

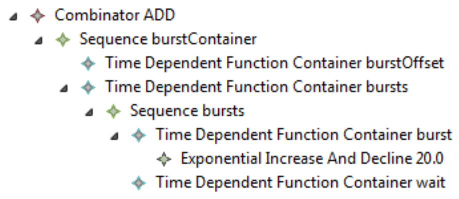


```
Combinator MULT
  Sequence trends
    Time Dependent Function Container trendOffset
      Constant 1.3333333333333333
    Time Dependent Function Container trend1
      Linear Trend 1.3333333333333333
    Time Dependent Function Container trend2
      Linear Trend 1.0
    Time Dependent Function Container holdLastValue
      Constant 0.8333333333333333
```

Figure 4.14.: A typical result of the *Trend Part*'s Model-to-Model transformation from hl-DLIM to DLIM.

4.2.5.3. Transforming the Burst Part

The *Burst* part is added to the root *Sequence* using and additive *Combinator*. It contains a *Sequence* with an offset *TimeDependentFunctionContainer* (for the *First Burst Offset*) and a second *TimeDependentFunctionContainer* containing another *Sequence* with the actual bursts.



```
Combinator ADD
  Sequence burstContainer
    Time Dependent Function Container burstOffset
    Time Dependent Function Container bursts
      Sequence bursts
        Time Dependent Function Container burst
          Exponential Increase And Decline 20.0
        Time Dependent Function Container wait
```

Figure 4.15.: A typical result of the *Burst Part*'s Model-to-Model transformation from hl-DLIM to DLIM.

This last *Sequence* contains two *TimeDependentFunctionContainers*, one with the actual *Burst Function*, and one offset container for the *Inter Burst Period*. This *Sequence* repeats for the model's entire duration.

4.2.5.4. Transforming the Noise Part

The *Noise Part* is transformed to an additive *Combinator* within the root *Sequence*. This *Combinator* contains a *UniformNoise* function. The function's *min* and *max* attributes are set using the *Minimum Noise Arrival Rate* and *Maximum Noise Arrival Rate* parameters.

4.3. Conclusions

This chapter introduces and describes the Descartes Load Intensity Model (DLIM), which can be used to describe load intensity variations over time. It does so by providing a way to structure piece-wise mathematical functions over a defined duration. Thus, the first of the research questions from Section 1.2 (Is it possible to create a model to capture load intensity variations over time?) is answered: It is possible to model load intensity variations over time using a structured approach for composing piece-wise mathematical functions.

This piece-wise function approach has a few drawbacks, however. It does not allow easy recognition of load intensity behavior at first glance, and the creation of custom load intensity variations can result in tedious work. This is addressed by the high-level Descartes Load Intensity Model (hl-DLIM), which uses a smaller set of parameters to condense the information about a load intensity variation. This should help to alleviate concerns when addressing research question 1.a (Can this model be used for custom load intensity variation creation for specific benchmarking purposes?). A model-to-model transformation transforms hl-DLIM instances into DLIM instances.

The accuracy and usability of these two models is extensively evaluated in Chapter 7.

5. LIMBO - The Descartes Load Intensity Modeling Platform

LIMBO is a DLIM modeling environment, which is based on the Eclipse Modeling Framework (EMF) [SBMP08]. Thus, it is a plug-in feature for the Eclipse IDE, which provides an editor for the creation and modification of load intensity models, as well as additional utilities for model application.

The DLIM editor has been automatically generated by EMF using an EMF genmodel and Ecore meta-model. Section 5.2 gives an outline of the plug-ins and code that are part of the entire feature. It also points out which parts of the code are generated and which have been added or modified manually.

The following sections shows the functionality of all other manual additions to the generated code base. These additions are split into three parts:

- **Model Evaluation**: These additions help to evaluate models and derive the described instances. This includes a core Model Evaluator (see Section 5.3) and the generators for arrival rate and request time-stamp series (see Section 5.4).
- **Modeling Process Assistance**: LIMBO provides assistance (such as the Model Creation Wizard 5.6), which help modelers to easily use LIMBO by providing structure and guidance during the process of modeling. Additionally, LIMBO provides help and automation for using the DLIM instance extraction processes, as defined in Chapter 6 .
- **Utilities**: Additional functionality that is useful when working with DLIM. This includes utilities, such as generation of arrival rate time-series from a time-stamp series (see 5.7.2), and a tool that calculates the difference between an arrival rate file and a model instance (see 5.7.1).

A tutorial for the use of LIMBO can be found in Appendix A.

5.1. Requirements

The functional requirements of LIMBO's DLIM tools are mostly derived from the goals of this thesis in general. They are:

- Creation and modification of models and their elements.

- Creation of arrival rate and request time-stamp series based on models.
- Assistance for following the DLIM modeling process.
- Validation feedback to the user.
- Ability to compare traces (arrival rate or time-stamp series) with model instances.

Non-functional requirements:

- Usage of the modeling environment shall require no extensive setup.
- Extensibility: The plug-ins should be easily extensible for the purposes of future work.
- The tools should demand as few processing resources (CPU, memory) as possible.
- Generated and hand-written code should be kept separate.

5.2. LIMBO Architecture

LIMBO consists of five plug-ins, as visualized in Fig. 5.1 and Fig. 5.2:

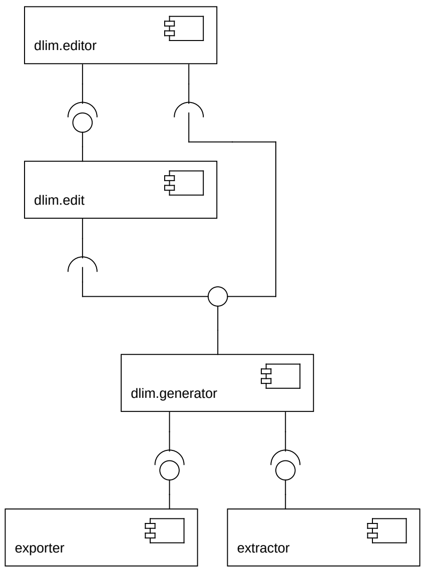
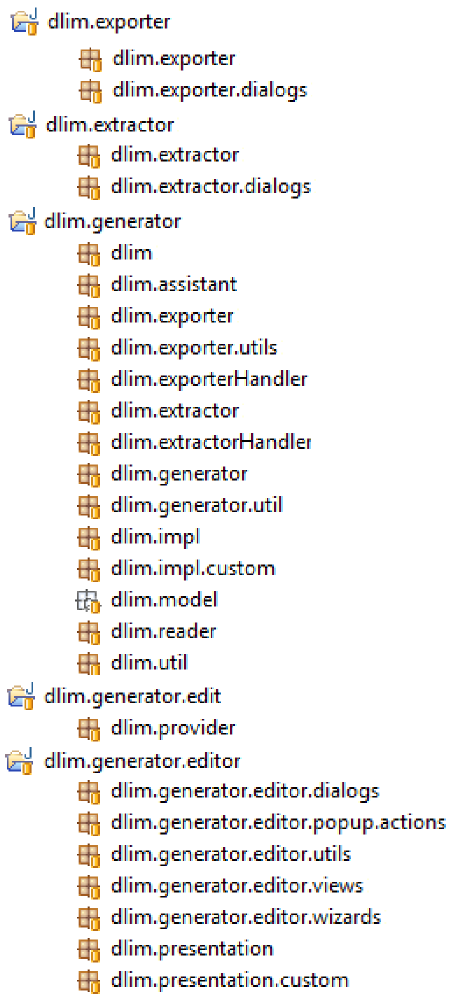


Figure 5.1.: LIMBO architecture.

- **dlim.generator**: Contains the meta-model, its implementation, and evaluation functionality.
- **dlim.generator.edit**: Contains the model element providers.
- **dlim.generator.editor**: Contains the editor and all other GUI elements, such as right-click menu elements.
- **dlim.exporter**: Contains default implementations for the Exporter extension point.
- **dlim.extractr**: Contains default implementations for the Extractor extension point.

The following sections describes the artifacts within these plug-ins, their functions, and elaborates on which artifacts are generated and which are hand-written.

Note, that all generated code is annotated with the *@generated* tag. Modifications within generated files are annotated using the *@generated not* tag. Completely hand-written files have none of those tags.



- dlim.exporter
  - dlim.exporter
  - dlim.exporter.dialogs
- dlim.extractor
  - dlim.extractor
  - dlim.extractor.dialogs
- dlim.generator
  - dlim
  - dlim.assistant
  - dlim.exporter
  - dlim.exporter.utils
  - dlim.exporterHandler
  - dlim.extractor
  - dlim.extractorHandler
  - dlim.generator
  - dlim.generator.util
  - dlim.impl
  - dlim.impl.custom
  - dlim.model
  - dlim.reader
  - dlim.util
- dlim.generator.edit
  - dlim.provider
- dlim.generator.editor
  - dlim.generator.editor.dialogs
  - dlim.generator.editor.popup.actions
  - dlim.generator.editor.utils
  - dlim.generator.editor.views
  - dlim.generator.editor.wizards
  - dlim.presentation
  - dlim.presentation.custom

Figure 5.2.: LIMBO's plug-ins and packages.

5.2.1. DLIM Generator Plug-in

The dlim.generator plug-in contains the meta-model Ecore file and the genmodel from which the default meta-model interfaces, their implementations, providers, and the editor are generated. It also contains the meta-model element interfaces and implementations, as well as their default utilities (such as validation utilities). It also contains the adapters that modify model element behavior, the model evaluation tools, and the arrival rate and time-stamp series generators (hence the plug-in's name).

It features two Extension Points:

- **Exporter**: Allows the addition of exporters that export a DLIM instance to a file. Exporters must implement the *dlim.exporter.IDlimExporter* interface.

  Default exporters are contained in the *dlim.exporter* plug-in.

- **Extractor**: Allows the addition of extractors, which extract a model instance from an existing trace. Extractors must implement *dlim.reader.IDlimArrivalRateReader* for their trace parser and *dlim.extractor.IDlimExtractor* for the model instance creator. Default implementations for both interfaces are provided, in case the user wants to implement only one of these interfaces.

  Default extractors are contained in the *dlim.extractor* plug-in.

The following packages within the *dlim.generator* plug-in are visible to other plug-ins:

- *dlim*
- *dlim.assistant*
- *dlim.exporter*
- *dlim.extractor*
- *dlim.generator*
- *dlim.reader*
- *dlim.util*

Two additional packages, *dlim.exporterHandler* and *dlim.extractorHandler* are only visible to the **dlim.generator.editor** plug-in.

The following packages are part of the *dlim.generator* plug-in:

5.2.1.1. dlim.model

This package contains the meta-model Ecore file and the generator model (genmodel) file, which EMF uses to generate the default editor and meta-model interfaces, implementations, and providers.

5.2.1.2. dlim

This package contains the model element interfaces. This package is generated by the EMF genmodel and has only one custom hand-written modification:

- *DlimFactory.eINSTANCE* has been modified to return a *CustomDlimFactoryImpl* as instance, instead of the generated *DlimFactoryImpl*.

5.2.1.3. dlim.impl

This package contains the default model element implementations. All instances are created by the *dlim.DlimFactory*, of which an implementation is provided as well. This package and its contents have been generated by the EMF genmodel. A few modifications have been made though:

- *Validation constraints*: The validation constraints (see 4.1.6) are part of the meta-model and are thus being generated by the genmodel. The originally generated methods contain errors that prohibit compilation. For this reason the custom validation constraint implementations have been implemented within the generated code.

5.2.1.4. dlim.util

This package contains the utilities generated by the genmodel, whereas *DlimAdapterFactory* and *DlimSwitch* are stubs without any functionality, *DlimValidator* contains the diagnostics tools that use the generated validation methods (see 4.1.6) for model validation. This code has no hand-written modifications or additions.

5.2.1.5. dlim.impl.custom

This package contains the *CustomDlimFactoryImpl*. This class inherits the *dlim.impl. DlimFactoryImpl*. It overrides the element creation methods for the elements for which custom implementations exist.

Also contained in this package are the custom model element implementations that add functionality not implemented in the generated implementations in *dlim.impl*. This allows the safe and easy separation of generated and hand-written code (with the exception of the validation methods).

At the moment, only *ArrivalRatesFromFile* (see 4.1.5.6) has a custom implementation. This custom implementation implements the methods described in Section 4.1.7.3.

5.2.1.6. dlim.exporter

This package contains the **IDlimExporter** interface, which must be implemented by any plug-in wishing to make use of the **Exporter** extension point.

5.2.1.7. dlim.exporter.utils

This package contains utilities that help when implementing a new exporter. The use of these utilities is highly recommended.

- **ArrivalRateGenerator**: Provides functionality to sample a list of arrival rates from a DLIM instance, represented by its *dlim.generator.ModelEvaluator*. It is also able to write this list to a file. See Section 5.4 for further details on arrival rate time series generation.
- **TimeStampWriter**: Provides functionality to write request time stamps using a list of arrival rates (as is provided by *ArrivalRateGenerator*. See Section 5.4.2 for further details on request time stamp generation.
- **DlimFileUtils**: Miscellaneous utilities for accessing DLIM files and handling their file paths and contents.

5.2.1.8. dlim.exporterHandler

This package is hidden to all plug-ins except for the *dlim.editor* plug-in. It manages all registered implementations of the **Exporter** extension point.

5.2.1.9. dlim.extractor

This package contains classes and interfaces used for the **Extractor** extension point.

- **IDlimExtractor**: This interface must be implemented by any extractor implementing the **Extractor** extension point.
- **SimpleExtracotr**: *IDlimExtractor* default implementation. This extractor should be used for testing custom arrival rate readers, as *dlim.reader.IDlimArrivalRateReader* must also be implemented when implementing the **Extractor** extension point.
- **ModelExtractor**: Provides static functionality for the automated parts of model extraction based on the model extraction processes defined in Chapter 6.
- **HLDlimParameterContainer**: A container for all parameters used in hl-DLIM. Use this when extracting hl-DLIM instances.

5.2.1.10. dlim.extractorHandler

This package is hidden to all plug-ins except for the *dlim.editor* plug-in. It manages all registered implementations of the **Extractor** extension point.

5.2.1.11. dlim.generator

This package contains the model evaluation logic, primarily used for arrival rate and time-stamp series generation.

The following classes provide the model evaluation logic:

- **ModelEvaluator**: The primary access point to all model evaluation logic. It is instantiated using the model's root element (which is always a *Sequence*) and a seed for the random number generator (for *Noise* evaluation).

  It provides the **getArrivalRateAtTime(double rootTime)** method, which returns the model's resulting arrival rate for a given time.

- **ModelEvaluatorUtil**: Provides additional utilities for model evaluation, such as the abilitiy to get the parent element within the models tree, or to get the duration of the *TimeDependentFunctionContainer* or *Sequence* that is the (possibly indirect) parent of a given *Function*.

These classes provide additional functionality, mostly used when generating time series:

- **DiffAnalyzer**: Uses a *ModelEvaluator* to sample the model at the points in time specified within the provided arrival rate .txt file. Returns the difference between the read file and the model instance arrival rates in a .txt file. Also returns statistical values about the overall differences. The *DiffAnalyzer*'s functionality is further described in Section 5.7.1.

- **ArrivalRateTuple**: Provides a class to store the tuple of arrival rates and their time-stamps. Also provides a method to get the sampling interval by getting the difference of two tuple's time-stamps.

- **IGeneratorConstants**: Provides constants for the *TimeStampGenerator*. These constants indicate the distribution of the time-stamps.

5.2.1.12. dlim.generator.util

This package contains utilities for the *ModelEvaluator*. This package is hidden from all plug-ins, with the exception of *dlim.generator*.

- **FunctionValueCalculator**: Contains the logic to evaluate the values for single simple functions (simple functions being the *Functions* that do not contain nested *Functions*). To do this, the *FunctionValueCalculator* reads the attributes of the passed concrete *Function* and returns its arrival rate for the given input value x (This value has been derived by the *ModelEvaluator*, using the current time and the *ReferenceClocks*).

- **TimeKeeper**: The *ModelEvaluator* uses the *TimeKeeper* to set the derived time attributes (see 4.1.7.1) for *Sequences* and *TimeDependentFunctionContainers*. Beginning at the root *Sequence*, the *TimeKeeper* descends the model's tree and sets the *firstIterationStart* and *firstIterationEnd* times, as well as the *loopDuration* and *finalDuration* for *Sequences*.

5.2.1.13. dlim.reader

This package contains classes and interfaces responsible for the parsing of time series.

- **ArrivalRateReader**: Provides functionality to read arrival rates from an arrival rate file. Can read either a single arrival rate at a given time, or returns a list of all *ArrivalRateTuples* contained in the file.

- **IDlimArrivalRateReader**: Interface for an arrival rate reader. Must be implemented by a reader for the **Extractor** extension point.

- **DefaultArrivalRateReader**: A default implementation of *IDlimArrivalRateReader*. Is able to read arrival rate files of the same format as produced by the arrival rate file exporter.
- **ReadingUtils**: Provides additional functionality for the reading of files (such as extracting the file name from its path).
- **RequestTimeSeriesReader**: Provides functionality to parse a request time-stamp trace into an file containing the arrival rates per second for each second. The reader's functionality is further described in Section 5.7.2.

5.2.1.14. dlim.assistant

This package contains modeling process assistant logic. It provides calibration utilities which can be used by the extraction process and the user. It contains:

- **Calibrator**: Provides functions for the calibration of the set values within interpolated functions. For a more detailed description of calibration functionality, see Section 5.5.
- **CalibrationException**: Is thrown by the *Calibrator* in case of a calibration error.

5.2.2. DLIM Generator-Edit Plug-in

This plug-in contains the providers used by the editor. These providers provide display specific information, such as the element's display images and labels. All classes in this plug-in have been generated by the EMF genmodel.

The following hand-written changes have been made to the generated code in the *dlim.provider* package:

- **TimeDependentFunctionContainerItemProvider:: addDurationPropertyDescriptor(Object object)**: The unit of seconds *(s)* has been added to the *duration* property description.
- **SequenceItemProvider::addTerminateAfterTimePropertyDescriptor(Object object)**: The unit of seconds *(s)* has been added to the *terminateAfterTime* property description.
- **PolynomialItemProvider::getText**: The editor label has been changed to represent the polynomial's contents (see Section 4.1.7.2).
- **PolynomialItemProvider::notifyChanged(Notification notification)**: Changes the editor label each time the polynomial's contents have been changed.

5.2.3. DLIM Generator-Editor Plug-in

The dlim.editor plug-in contains all GUI elements and their utilities. It is organized into the following packages:

5.2.3.1. dlim.presentation

This package contains all editor classes generated by the EMF genmodel. They are:

- **DlimEditorPlugin**: The plug-in singleton.
- **DlimEditor**: The editor.
- **DlimActionBarContributor**: Contributes generic actions (such as load, save) to the toolbars and menus.
- **DlimModelWizard**: The default wizard for the creation of a new model instance.

All these classes are completely generated and unmodified.

5.2.3.2. dlim.presentation.custom

Contains the **CustomDlimEditor**, which is a custom implementation of the *DlimEditor*. The changes in the custom editor are geared towards updating the plot view when changes have been made.

5.2.3.3. dlim.generator.editor.views

This package contains the **PlotView**, which is an Eclipse view that contains a **PlotCanvas** on which the arrival rate function that results from the current DLIM instance is displayed.

5.2.3.4. dlim.editor.wizards

This package contains the **CustomDlimModelWizard**. This wizard subclasses the generated *dlim.presentation.DlimModelWizard*. It provides a custom hand-written wizard for the creation of model instances. This wizard prompts the user for the parameters defined as part of hl-DLIM (see Section 4.2) and then creates a new DLIM instance.

It also contains all the wizard pages used by the *CustomDlimWizard*, they are:

- **DlimPageChoiceModelWizardPage**: This page allows the user to choose which pages are to be displayed later in the wizard and which pages are to be omitted.
- **DlimModelWizardPage**: An abstract wizard page that contains a plot canvas, which visualizes the current DLIM instance. The interactive wizard page area has to be filled by its child implementations.
- **DlimReadFileModelWizardPage**: Allows the user to extract the hl-DLIM parameters from an existing arrival rate trace file.
- **DlimSeasonalModelWizardPage**: Prompts the user for the paramters of the hl-DLIM's *Seasonal Part*.
- **DlimTrendModelWizardPage**: Prompts the user for the parameters of the hl-DLIM's *Trend Part*.
- **DlimBurstModelWizardPage**: Prompts the user for the parameters of the hl-DLIM's *Burst Part* and *Noise Part*.

5.2.3.5. dlim.generator.editor.popup.actions

Contains all the right-click context menu elements added by LIMBO. They are:

- **TimeStampGeneratorAction**: Allows the user to select an exporter (implementing the **Exporter** extension point in the *dlim.generator* plugin). This exporter is then provided with a *dlim.generator.ModelEvaluator* for the selected DLIM instance and executed.
- **TimeSeriesReaderAction**: Reads a .txt time-stamp series into an arrival rate series. Uses *dlim.reader.RequestTimeSeriesReader*.
- **DiffRunnerAction**: Calculates the difference between the arrival rates within the model and a given arrival rate series (.txt file). Prompts the user for the .txt file and random number generator seed using a *dlim.generator.editor.dialogs.LaunchDiffDialog*. It then initializes a *dlim.generator.ModelEvaluator* and passes it to the *dlim.generator.DiffAnalyzer*.
- **CalibrationAction**: This abstract action contains the abstract **executeCalibration(double desiredValue)** method, which launches the *dlim.assistant.Calibrator* for the respective attribute in the respective model element. Its concrete children are:

  - **CalibrateTrendStartValue**: Calibrates the *startValue* attribute of a *Trend*, so that the model output at that time equals a desired value.
  - **CalibrateTrendEndValue**: Calibrates the *endValue* attribute of a *Trend*, so that the model output at that time equals a desired value.
  - **CalibrateBurstPeakValue**: Calibrates the *peakValue* attribute of a *Burst*, so that the model output at that time equals a desired value.

- **DecomposeInPlotViewAction**: Decomposes the selected *Sequence* in the *PlotView*.
- **ExtractionAction**: Allows the user to select an extractor (implementing the **Extractor** extension point in the *dlim.generator* plugin). This extractor is then executed.
- **OpenPlotViewAction**: Opens the *PlotView*.
- **PlotArrivalRateFileInPlotViewAction**: Plots an arrival rate trace file in the *PlotView* for comparison.
- **SaveGraphFromPlotViewAction**: Saves the graph that is currently displayed in the *PlotView* to a .png file.
- **ToggleDecomposeInPlotViewAction**: Toggles the *PlotView* decomposition visualization (see Section5.7.3).

5.2.3.6. dlim.generator.editor.dialogs

This package provides the dialogs that prompt the user for parameters when running an action (see *dlim.generator.editor.popup.actions*). They are:

- **LaunchDiffDialog**: Prompts the user for the .txt arrival rate series file that is to be compared with the model. Also asks for the random number generator seed for model evaluation.
- **DiffResultsDialog**: Displays the results of the difference calculation between a DLIM instance and an arrival rate trace file.
- **PlotArrivalRateFileDialog**: Prompts the user for parameters in order to plot the arrival rates from an arrival rate trace file in the *PlotView*.
- **SavePlotViewImageDialog**: Prompts the user for parameters in order to save the contents of the *PlotView* to a file.
- **SelectExporterDialog**: Displays all registered implementations of the **Exporter** extension point and lets the user pick one to use.
- **SelectExtractorDialog**: Displays all registered implementations of the **Extractor** extension point and lets the user pick one to use.
- **TimeStampGeneratorParametersDialog**: Prompts the user for the parameters for arrival rate and request time-stamp series generation.
- **LaunchCalibrationDialog**: Prompts the user for the desired value to calibrate to. Alternatively uses *dlim.reader.ArrivalRateReader* to read this value from an arrival rate file. Then launches *CalibrationAction.executeCalibration(desiredValue)*. Errors in form of a *dlim.assistant.CalibrationException* are displayed to the user.

5.2.3.7. dlim.generator.editor.utils

Contains utilities that help the GUI elements to deal with the Eclipse platform. Contains only the **ProjectManager** at the moment. The *ProjectManager* helps with the management of the current Eclipse Project, its resources and selections within this project.

5.2.4. DLIM Exporter Plug-in

The dlim.exporter plug-in offers default implementations of the *dlim.generator* plug-in's *dlim.exporter.IDlimExporter* interface and the **Exporter** extension point.

Its three exporter implementations are contained in its *dlim.exporter* package:

- **DlimArrivalRateExporter**: Exports an arrival rate time series. Uses the *dlim.exporter.utils.ArrivalRateGenerator*.
- **DlimEqualDistanceRequestStampExporter**: Exports request time stamps with an equal distance from one another within each sampled arrival rate interval. Uses the *dlim.exporter.utils.TimeStampWriter*.
- **DlimUniformRequestStampExporter**: Exports request time stamps with a uniform random sampling within each sampled arrival rate interval. Uses the *dlim.exporter.utils.TimeStampWriter*

The *dlim.exporter.dialogs* package contains the dialogs displayed to the user for exporter parameters.

5.2.5. DLIM Extractor Plug-in

The dlim.extractractor plug-in offers default implementations of the *dlim.generator* plugin's *dlim.extractor.IDlimExtractor* interface and the **Extractor** extension point.

Its two extractor implementations are contained in the *dlim.extractor* package, both use dialogs in te *dlim.exporter.dialogs* package for user parameter input:

- **PeriodicProcessExtractor**: Extracts a DLIM model instance based on the periodic extraction process. Uses the *dlim.extractor.ModelExtractor*.
- **SimpleProcessExtractor**: Extracts a DLIM model instance based on the simple extraction process. Uses the *dlim.extractor.ModelExtractor*.

Both **Extractor** extension point implementations in this plugin use the provided default *dlim.reader.ArrivalRateReader* arrival rate reader provided by the *dlim.generator* plugin.

5.3. DLIM Evaluator

The DLIM Evaluator reads the Model element attributes and provides the functionality of being able to calculate the model's arrival rate for a set point in time.

To achieve this, the evaluator must first set the derived time values described in Section 4.1.7.1 during initialization. These values are calculated from the *Sequence* and *TimeDependentFunctionContainer duration* and *loops* attributes. At this time, other model initialization methods, such as *ArrivalRatesFromFile*'s *readFile()* (see 4.1.7.3) method are also called. The evaluator also initializes the random number generator, used for *Noises* with a set seed provided by the user. This seed makes the evaluation result reproducible.

When calculating the arrival rate for a certain point in time, the evaluator then checks the current time against the start and end times of the *Sequences* and *TimeDependentFunctionContainers*, to see which are currently active. The running *TimeDependentFunctionContainer*'s functions are then evaluated. All inactive *Sequences* and *TimeDependentFunctionContainers* return 0 (or 1, depending on the parent Combinator).

Since *Sequences* and *TimeDependentFunctionContainers* are held by parent *Sequences*, which can loop, they can also start and end at multiple points in time (A start and end time during the first loop iteration, one during the second and so on). To account for this, the time used for checking if a *TimeDependentFunctionContainer* is still active (called *guard time*), is always the time during the first loop of the *Sequence* holding the *TimeDependentFunctionContainer* (i.e. $guardTime = Sequence :: start + loopTime$). When calculating this *guard time*, the current time references (*loop time* and *seq time*) are also calculated, so that they can be stored in the *ReferenceClockObjects* and used as function inputs (for more on *Reference Clocks* see Section 4.1.3.1).

The model evaluator also provides all of the logic necessary to calculate the function values from the model element attributes. This logic is not included in the model element implementations themselves for two major reasons:

1. The model element implementations are generated by the EMF genmodel:

   In order to keep generated and hand-written code separated, almost no evaluation logic is included in the model element implementations. The only exception is:

   - **ArrivalRatesFromFile::getArrivalRate(EDouble x)**: This method returns the interpolated arrival rate from the arrival rate ArrayList stored within the model element (see 4.1.7.3).

2. Increased code-re-usability:

   Since *Bursts* logically consist of two *Trends* (one increasing trend until peak is reached, and one decreasing trend afterward), it is good practice to reuse *Trend* evaluation logic for *Bursts*, if possible. Keeping parametrized function evaluation methods independent from the actual model elements enables their reuse in different contexts.

5.4. Arrival Rate and Request Time-Stamp Series Generator

The DLIM plug-in features two separate time series generators. The arrival rate series generator generates a time series of arrival rates with their time stamps, whereas the request time-stamp series generator generates pure request/user arrival time-stamps based on the arrival rates provided by the arrival rate series generator.

5.4.1. Arrival Rates Series Generator

The arrival rate generator uses the *Model Evaluator* to calculate the arrival rates at given points in time. It samples the arrival rates at a **sampling interval** provided by the user. It starts at $samplinginterval/2$ (this is useful for the time-stamp generator, see 5.4.2) and continues until the current time is greater or equal to the *finalDuration* of the root *Sequence*.

Note that the width of the sampling interval has a significant impact on the final result. Under-sampling causes loss of details, whereas over-sampling may lead to rounding errors during time-stamp generation. *Noises* are also only evaluated at the points at which they are sampled. This leads to the sampling interval having a significant impact on the resulting *Noise*.

5.4.2. Request Time-Stamp Series Generator

The request time-stamp generator creates request/user arrival time-stamps based on the arrival rates provided by the arrival rate generator.

It does so by creating the time-stamps within an interval around the time-stamp of the arrival rate. This interval has the same width as the sampling width of the arrival rate generator. (This is the reason why the first arrival rate is sampled at $samplinginterval/2$. The first time-stamp creation interval is then $[0, samplinginterval)$.) Centering the time-stamp generation interval around the measured point of the arrival rate is intended to minimize the overall error that results from the sampling.

The request time-stamp generator provides two methods with which to generate the time-stamps within this interval:

- **Equal Distance**: The time-stamps are spread throughout the interval with equal distances.
- **Uniform Random Distribution**: Time-stamps between $begin_{interval}$ and $end_{interval}$ are generated using a uniform random distribution.

The amount of time-stamps generated within each interval is $arrivalrate * intervalwidth$. The time-stamp generator provides an **arrival rate devisor**, which can be used to change the number of time-stamps. The resulting amount of time-stamps within the interval is then

$$\frac{arrivalrate*intervalwidth}{arrivalratedevisor}.$$

This is especially useful when creating "mini-benchmarks" from models based on huge cloud or web systems with arrival rates that order in the millions.

The **time stretch** parameter also helps in this scenario. It can be used to compress or stretch the time used for time-stamp generation in relation to the time used for arrival rate generation. This may lead to an amplification of sampling errors, however. As a result, the sampling interval width should always be set accordingly. The property of *self similarity* (see Section 2.4) holds true when stretching the time scale, since the arrival rates are sampled from the original arrival rate function within the unscaled model instance.

The last parameter used to set up the request time-stamp generator is the **amount of decimal places** a time-stamp may have. This accounts for the fact that benchmarking frameworks (which the time-stamp series is usually intended as input for) only have a limited accuracy. By limiting the amount of decimal places, the user can account for this limitation.

5.4.2.1. Errors due to Sampling

Under-sampling may lead to problems with signal reconstruction in general. In the context of the time-series generator, these problems may be enhanced, specifically when using the random distribution for time-stamp generation. The random distribution may lead to a lopsided distribution of time-stamps within the sampling interval. This may lead to arrival rates that go contrary to the unsampled arrival rates of the original model.

Over-sampling problems lead to errors due to the fact that the request time-stamp generator always creates an integer amount of time-stamps, whereas the arrival rates provided are floating point values. The decimal places of the arrival rates must either be rounded to the next integer number or discarded (rounding to the lower integer number). While this leads to an error in itself, this error is enhanced by over-sampling. Considering the amount of time-stamps is calculated by: $\frac{arrivalrate * intervalwidth}{arrivalratedevisor}$, a smaller interval width leads to a greater significance of the decimal places of the resulting number of time-stamps. Rounding this number can add or discard time-stamps a maximum of one time-stamp per interval. A greater amount of intervals leads to a greater amount of discarded or erroneously added time-stamps.

Consider a constant function with an arrival rate value of 1 as an example. When sampling this function with an interval width of 1, the amount of time-stamps in this interval is: $arrivalrate * intervalwidth = 1.0 * 1.0 = 1.0 \approx 1$. Sampling the same function with a smaller sampling interval of $\frac{1}{3}$ however results in the loss of the time-stamp since the amount of time-stamps is now: $arrivalrate * intervalwidth = 1.0 * \frac{1}{3} = \frac{1}{3} \approx 0$ for each interval, resulting in 0 time-stamps for all three intervals covering the same interval as the original interval with a width of 1.

5.5. Trend and Burst Calibration

Trends and *Bursts* are interpolated functions that interpolate between their attributes (*Trend* from *startValue* to *endValue*, *Burst* from *baseLevel* to *peakValue* and then back to *baseLevel*). Since they reach these defined values at defined times, it is easy to automatically set these values in a way that results in the entire model producing a desired total arrival rate (*desiredValue*).

The model's total arrival rate derives directly from the *Trend*'s or *Burst*'s unknown arrival rate. it is only modified by the output of all other concurrently running *Functions*. Since all concurrently running functions are connected via *Combinators*, these other *Functions* can only be added, multiplied, and subtracted from the unknown attribute, which is being calibrated. Since the point in time at which calibration occurs is also a set and a constant point in time, it follows that all *Function* output values are also constant (Note: When calibrating, all *Noises* are set to 0). As a result, the model output can be described this way:

$$desiredValue = constantLevel + constantFactor * unknownAttributeValue$$

desiredValue is set by the user. *constantLevel* and *constantFactor* can be easily extrapolated, by evaluating the model with two set values for the attribute which is being calibrated. The Calibrator thus evaluates the model twice, with the attribute being set to 0 the first, and being set to 1 the second time. *constantLevel* and *constantFactor* can now be derived:

$$constantLevel = modelEvaluatonResult(attribute = 0)$$

$$constantFactor =$$
$$modelEvaluatonResult(attribute = 1) - modelEvaluatonResult(attribute = 0)$$

Once these are known the attribute's value can simply be calculated:

$$unknownAttributeValue = \frac{desiredValue - constantLevel}{constantFactor}$$
$$= \frac{desiredValue - modelEvaluatonResult(attribute=0)}{modelEvaluatonResult(attribute=1) - modelEvaluatonResult(attribute=0)}$$

5.5.1. Calibration Errors

Two error types can occur during calibration:

- **Calibration results in 0**:

  $modelEvaluatonResult(attribute = 1) - modelEvaluatonResult(attribute = 0) = 0$

  This usually occurs, if the *Function* that is being calibrated is not being executed at the point in time at which the calibrated attribute is defined. In this case the change of the attribute has no effect on the model output at this time, making this method of calibration unusable.

  This effect always occurs when calibrating *Trend.endValue*, since the ModelEvaluator will already be evaluating the starting value of the next *TimeDependentFunction-Container*'s *Functions* at this point in time. In order to work around this problem, the point in time for which *Trend.endValue* is being calibrated is decremented by the minimum decrement for its *double* value, thus forcing model evaluation at a point of time, at which the *Trend* that is being calibrated is actually being executed. Tests have shown that the resulting error is reliably less then $1.0 * 10^{-14}$ and thus within acceptable boundaries.

- ***desiredValue* can not be reached**: This happens especially if the calibrated *Function* is a child of an *AbsoluteValueFunction* and *desiredValue* is negative.

The Calibrator tests for both of these errors. If they remain unresolved, a *CalibrationEx-ception* is thrown, and its message is displayed to the user.

5.6. Model Creation Wizard

When creating a new DLIM instance, LIMBO uses a model creation wizard for easy initial model creation. This wizard prompts the user for parameters, as defined by the high-level Descartes Load Intensity Model (hl-DLIM) (see Section 4.2). Each model part (*Seasonal Part*, *Trend Part*, *Burst Part*, and *Noise Part*) is queried in its own wizard page. The wizard also provides a method of extracting the hl-DLIM parameters from an existing arrival rate trace (see Section 6.5).

5.7. Additional Utilities

These utilities provide functionality that is useful to a modeler, without directly contributing to the goals of this thesis.

5.7.1. Difference Calculator

The Difference Calculator computes the difference between an arrival rate time series and a model instance. Just like the arrival rate and time-stamp series generators, it uses a pre-initialized Model Evaluator. It compares the arrival rate values within the file with the model evaluation result for every point in time specified in the read arrival rate file.

The Difference Calculator only takes an arrival rate series as input for comparison. A time-stamp series must first be converted using the provided Time-Series Reader (see 5.7.2).

The resulting difference between these two arrival rate values at the respective point in time, as read from the arrival rate trace, is printed to a .txt file. The difference calculator also returns the median and mean absolute difference as well as the relative median and mean differences. It also returns a difference metric based on DTW (see Section 7.1.2.2).

5.7.2. Time-Series Reader

The Time-Series Reader reads a time-stamp series and generates an arrival rate series from it. This arrival rate series can then be used as input for the Difference Calculator (see 5.7.1) or *ArrivalRatesFromFile* (see 4.1.5.6).

To do this, the Time-Series Reader counts the amount of time-stamps within each time bucket with the width of one second. It then prints this amount of time-stamps per second as the arrival rate at the bucket's "center" time (meaning $begin_{bucket} - length_{bucket}/2$).

5.7.3. Plot View

LIMBO offers a view to display the arrival rate function described by the current DLIM instance. This view also offers the functionality of displaying the arrival rates of an arrival rate trace for comparison and offers to visualize the impacts of *TimeDependentFunction-Containers* and *Combinators* on the total output function of their containing *Sequence*.

5.7.3.1. Combinator Impact Visualization

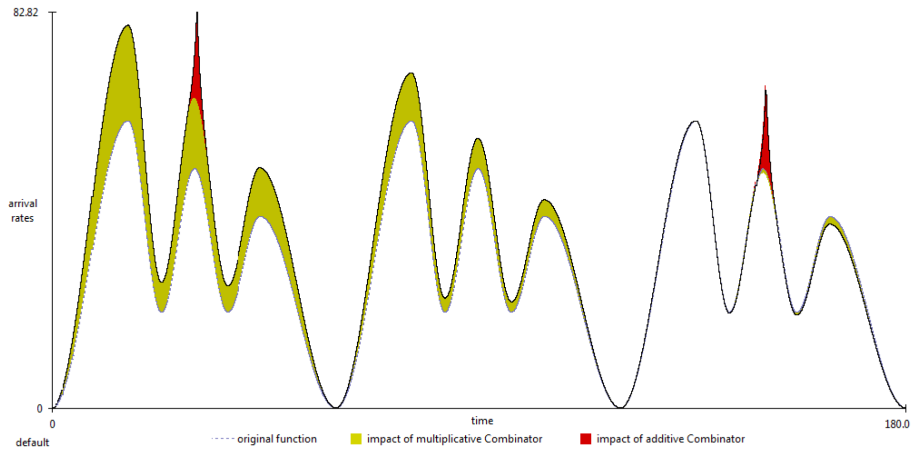


Figure 5.3.: An example visualization of *Combinator* impacts within a *Sequence*.

The *Combinator Impact Visualization* (also called *Sequence decomposition*), displays the impact of all *Combinators* contained within the *Sequence* in the Plot View. It does this by calculating the difference of the *Sequence*'s arrival rate function with and without the respective *Combinator*. This difference is then drawn as a colored area in the Plot View. A yellow area visualizes the impact of a multiplicative *Combinator*, whereas a red area visualizes the impact of an additive *Combinator*.

The *Sequence*'s function without any *Combinators* is displayed with a dashed blue line, whereas the final arrival rate function is displayed as a solid black line.

A colored area below the blue line of the original function signifies a negative impact (meaning that the corresponding *Combinator*'s function decreases the *Sequence*'s total function), whereas a colored area above the dashed blue line of the original function signifies a positive impact on behalf of the respective *Combinator*.

5.8. Conclusions

This chapter introduces LIMBO. LIMBO is the tooling platform that is being used for the modeling of DLIM instances and hl-DLIM instances. LIMBO's architecture as well as the functionality of its core components is discussed. Extensibility of LIMBO offers easy addition of functionality as part of future work.

6. Model Instance Extraction Process

This chapter presents automated methods for the extraction of Descartes Load Intensity Model (DLIM) instances from arrival rate traces. Each method requires a set of configuration parameters for the remaining automated model instance extraction. Some of the steps of the methods could be easily performed by hand, others require automation in order to be completed within a reasonable time frame.

Complete automation of the model instance extraction process however will be part of future work.

I define the following three methods:

1. **Simple Model Instance Extraction Process** (S-MIEP): Extracts a DLIM instance. This process (and its resulting DLIM instance) are inspired from the time-series decomposition approach in BFAST [VHNC10].

   The Simple Model Instance Extraction Process (S-MIEP) extracts a repeating *Seasonal Part* and a non-repeating *Trend Part*. This non-repeating *Trend Part* contains a list of *Trend* segments of fixed size, that interpolate between their start and end arrival rate value. The *Trend* list extends throughout the entire time duration for which the extracted model is defined. Additionally a *Burst Part* and an optional *Noise Part* are extracted. S-MIEP is visualized in Fig. 6.1.

2. **Periodic Model Instance Extraction Process** (P-MIEP): This is a variation of the simple extraction process that features multiple repeating trends. Again a DLIM instance is extracted, however, in contrast to S-MIEP, Periodic Model Instance Extraction Process (P-MIEP) does not feature a single list of equal length *Trend* segments. Instead it features multiple lists of *Trends*, each containing a fixed number of *Trend* segments of (potentially) different lengths. These *Trend* lists repeat and do not extend indefinitely as the single S-MIEP *Trend* list does. Examples for this are: Weekly trends and monthly trends.

3. **High-level Model Instance Extraction Process** (hl-MIEP): Extracts a hl-DLIM Instance. This process is based on the simple model extraction process. It uses the same information that is extracted by S-MIEP in order to fill the parameters defined by the high-level Descartes Load Intensity Model (hl-DLIM) (see Section 4.2).

Some parts of the model extraction process mirror the model-to-model transformation from a custom hl-DLIM instance to a DLIM instance (see Section 4.2.5), in that the extracted parameters are at times identical to the parameters provided by hl-DLIM. In these cases it follows that the creation of the DLIM instance is then identical to the model-to-model transformation. For instance, the *Trend* extraction extracts the exact same *Trend* list as the *Trend* list that defines the hl-DLIM *Trend Part*.

6.1. Extracting the Seasonal Part

The *Seasonal Part* of the read arrival rate trace is modeled using a *Sequence* of *TimeDependentFunctionContainers* and their *Functions*. This mirrors the approach taken in the model-to-model transformation in Section 4.2.5. Each *Function* within the *TimeDependentFunctionContainers* interpolates the corresponding peaks and lows within each seasonal period. As a result, the following data needs to be derived in order to build the *Seasonal Part*:

- The duration of a seasonal period
- Arrival rate peaks and their time-stamps
- Arrival rate lows and their time-stamps
- The function type used to interpolate between peaks and lows.

The seasonal period (length) is set by the user. It is usually selected using meta-knowledge about the trace. A trace that extends for multiple days, for example, has its period set as the duration of one day.

The peaks and lows are automatically determined by using a local minimum/maximum search on the arrival rates within the trace. The local arrival rate minima and maxima and their corresponding time-stamps within a seasonal period constitute the peaks and lows. Since the trace usually contains multiple seasonal periods, the respective median arrival rate value is selected for each local maximum and minimum within the *Seasonal Part*. Selecting the median instead of the mean reduces the impact of outliers on the extracted value. As a result, the derived functions interpolate first between the first median low and the first median peak, then between the first median peak and the second median low, and so on.

The last low must be of the same arrival rate as the first low in order for the *Seasonal Part* to repeat seamlessly.

The shape of the interpolating function (linear, exponential, logarithmic, sin) is selected by the user based on a best effort guess. Advanced heuristics for selection should be part of future work. According to my experience, the sin interpolation usually results in a good fit.

6.2. Extracting the Trend Part

The *Trend Part* consist of a series of functions (trend segments) that are either added or multiplied onto the *Seasonal Part* (the hl-DLIM-to-DLIM transformation in Section 4.2.5 works identically).

Each trend segment begins at the maximum peak of the *Seasonal Part* and ends at the maximum peak of the *Seasonal Part* in a later *Seasonal* iteration. This minimizes errors with trend calibration. The trend extraction uses the calibrator introduced in Section 5.5 in order to calibrate the trend in a way that the model output arrival rate at the trend segment's beginning (or end) equals the trace's actual arrival rate at the respective point in time.

The shape of the trend function (linear, exponential, logarithmic, sin) is selected using a best effort guess, similar to the selection of the seasonal shape.

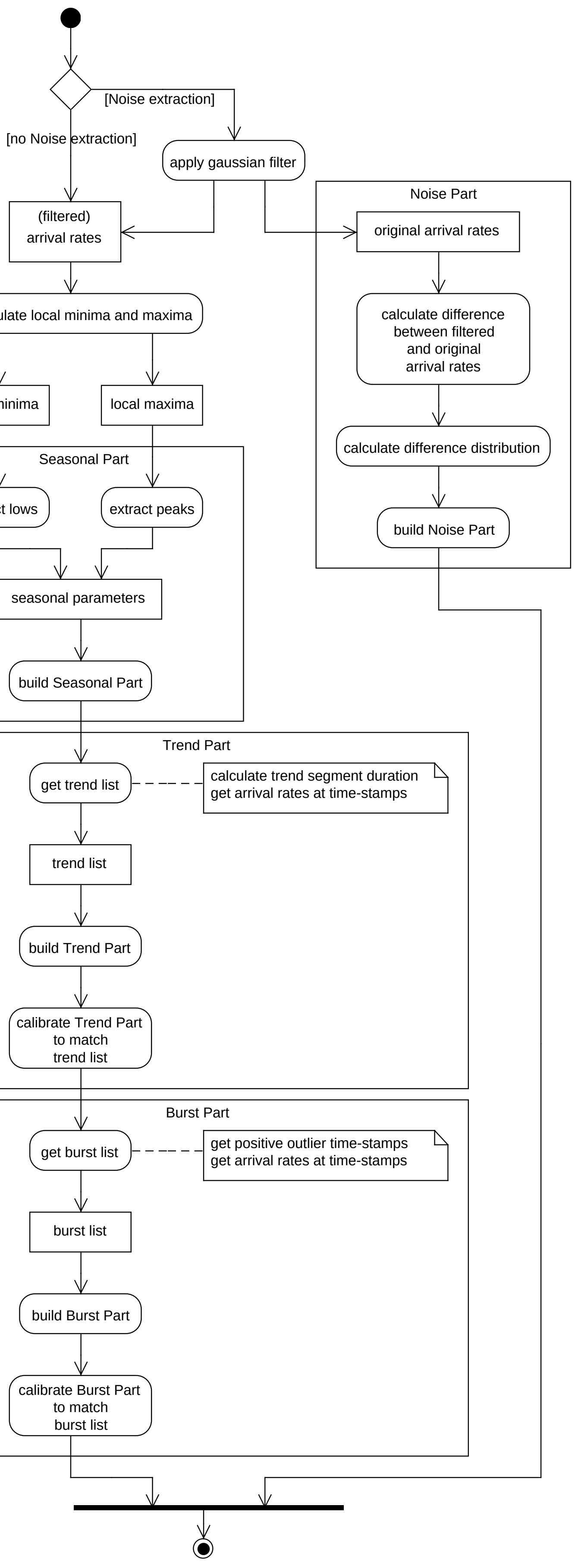

[Noise extraction]
[no Noise extraction]
apply gaussian filter
(filtered) arrival rates
Noise Part
original arrival rates
calculate difference between filtered and original arrival rates
calculate difference distribution
build Noise Part
calculate local minima and maxima
local minima
local maxima
Seasonal Part
extract lows
extract peaks
seasonal parameters
build Seasonal Part
Trend Part
get trend list
calculate trend segment duration
get arrival rates at time-stamps
trend list
build Trend Part
calibrate Trend Part to match trend list
Burst Part
get burst list
get positive outlier time-stamps
get arrival rates at time-stamps
burst list
build Burst Part
calibrate Burst Part to match burst list

Figure 6.1.: Activity Diagram of the *Simple Model Instance Extraction Process*.

6.2.1. Trend Part for S-MIEP

The simple extraction process features a list of equal-length trend segments. These segments have a set duration that is a multiple of the seasonal period. Just like the seasonal period, it is also usually selected using meta knowledge about the trace. These segments are then calibrated at their beginning and end to match the arrival rates in the trace.

6.2.2. Trend Part for P-MIEP

The periodic extraction process takes into account, that multiple repeating trends may be part of the arrival rate trace. Examples are weekly and monthly trends.

Since repeating trends (like the *Seasonal Part*) should end on the same arrival rate as the arrival rate they started on (allowing seamless repetition), each of these repeating trends contains at least two trend segments. These trend segments' duration is a multiple of the seasonal period. Unlike the trend segments in the simple extraction process they are not required to be of equal length, thus allowing odd multiples of seasonal periods as total trend durations.

The user selects lists of at least two trend segment durations for each repeating *Trend Part*. LIMBO's user interface currently (versions 0.14.3 and older) only allows lists of size 2 in order to minimize confusion in the user interface. P-MIEP and the automated extractor do however support longer lists.

6.3. Extracting the Burst Part

Extracting bursts is the matter of finding the points in time at which significant outliers from the previously extracted *Seasonal* and *Trend* parts are observed in the trace. Once a burst is found, it is added to the root Sequence and then calibrated to match the arrival rate from the trace.

Finding a burst requires the arrival rate in the trace to differ significantly from the predicted value based on the *Seasonal* and *Trend* parts. In order to eliminate false positives due to *Seasonal Parts* that are offset time-wise, the *Seasonal Part* used for the reference model in the burst recognition activity differs from the actual extracted *Seasonal Part*. The difference is that the *Seasonal Part* used in the burst recognition activity does not interpolate between the peaks and lows of the original arrival rate trace. Instead it interpolates only between the peaks. This removes false positives due to seasonal periods that are slightly offset in time, it does however also eliminate bursts that do not exceed the current seasonal peak. This trade-off is considered acceptable, since time-wise offset seasonal periods are commonly observed.

6.4. Extracting the Noise Part

The extraction of the *Noise Part* consists of two steps:

1. Noise reduction
2. Calculation of noise distribution

The idea behind this approach is to first reduce noise observed within the arrival rates contained in the trace, and then reconstruct the reduced noise by calculating the difference between the original trace and the filtered one.

Having filtered the noise, the extraction of the *Seasonal Part*, *Trend Part*, and *Burst Part* are then performed on the filtered trace. This has a significant impact on the extraction accuracy of these parts, and thus the accuracy of the extracted model instance, especially when extracting hl-DLIM instances, as shown in the model accuracy evaluation (see Section 7.1). Depending on the trace, even the total accuracy of DLIM extraction can be improved by noise elimination. In this case I recommend applying noise extraction, even if the extracted noise component itself is deleted later on.

6.4.1. Noise Reduction

Noise is reduced via the application of a one dimensional Gaussian filter on the read arrival rates.

A Gaussian filter has a kernel based on the Gaussian distribution, it thus has the following form (as defined in [BZ86]):

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

The width of a Gaussian filter's kernel is defined in relation to the standard deviation:

$$KernelWidth = 6\sigma - 1$$

I choose the kernel width depending on the *Seasonal* period and the expected number of peaks within a *Seasonal* period:

$$KernelWidth = \frac{SeasonalPeriod}{ExpectedMaximumSeasonalPeaks}$$

As a result the standard deviation is defined as:

$$\sigma = \frac{\frac{SeasonalPeriod}{ExpectedMaximumSeasonalPeaks} + 1}{6}$$

The frequency response of a Gaussian filter is:

$$\hat{g}(f) = e^{-\frac{f^2}{2\sigma_f^2}}, \text{ with } \sigma_f = \frac{1}{2\pi\sigma}$$

As a result any frequency higher than the expected maximum number of peaks will be smoothed.

The Gaussian filter is a quick and easy way of reducing high frequencies and thus reducing noise in a trace. Other more sophisticated noise filtering and anomaly detection methods, such as proposed in [Bie12], could be applied at this point.

6.4.2. Calculating the Noise Distribution

The *Noise Part* is modeled as a normally distributed random variable. This variable is added onto the DLIM instance's root *Sequence*.

The normal distribution's mean and standard deviation are calculated as the mean and standard deviation of the differences between the filtered arrival rate trace.

S-MIEP and P-MIEP both only support the extraction of normally distributed noise. Other noise distributions are not supported. The high-level Model Instance Extraction Process (hl-MIEP), however, supports the extraction of uniformly distributed noise.

6.5. Extracting a high-level Descartes Load Intensity Model Instance

The *high-level Model Instance Extraction Process* (hl-MIEP) is similar to the *Simple Model Instance Extraction Process.* This section only highlights the differences between those two processes.

6.5.1. Seasonal Part

Hl-DLIM is restricted to only support peaks with an equal distance from one another. The arrival rates of such peaks are linearly interpolated between the first peak's arrival rate and the last peak's arrival rate.

When extracting an hl-DLIM instance from an arrival rate trace, the difference thus lies in the *interval containing peaks* and in the search for the maximum and minimum peak.

The *interval containing peaks* is calculated as the time difference between the first and the last peak, the *first peak*'s arrival rate is then set either as the minimum or maximum peak (depending on whether the first median peak has a greater or a smaller arrival rate than the last median peak in the trace) and the *last peak* is set as the corresponding counter-part.

6.5.2. Trend Part

Extracting the *Trend Part* is done almost identically as in the simple model extraction process, since hl-DLIM defines its *Trend Part* as a list of arrival rates at the beginning and end of each trend segment, identically to the arrival rate list extracted in S-MIEP.

The only difference is the offset before the first trend segment begins. The trend segment always ranges from the maximum peak within one seasonal period to the maximum peak within a following seasonal period. The simple model extraction process allows this maximum peak to be any seasonal peak. Hl-DLIM, however, only allows the first or last peak to be the maximum peak. As a result the time offset for the first trend segment is slightly different.

6.5.3. Burst Part

Bursts are detected and calibrated using the same peak-only *Seasonal Part* as in Section 6.3.

While the other model extraction processes modeled each burst individually, hl-DLIM only supports recurring bursts. Thus, only the *first burst offset* and the *inter burst period* are extracted, as well as only a single *burst arrival rate.*

The first burst offset is selected based on its time-stamp, whereas the period between recurring bursts is calculated as the median *inter burst period* from the independent bursts. The burst arrival rate is also calculated as the median burst arrival rate.

6.5.4. Noise Part

Hl-DLIM Noise is extracted using the filtering approach, thus having the same noise reduction side-effects as in the other model extraction processes.

Hl-DLIM, however, only supports a uniformly distributed random variable as noise. In order to eliminate outliers, the minimum and maximum value of the respective uniform distribution are selected as the 10^{th} and 90^{th} percentile of the difference between the filtered and unfiltered arrival rates.

6.6. Conclusions

This chapter presents three model extraction methods that allow the extraction of model instances from an arrival rate trace, thus answering research question 2 from Section 1.2 (Is it possible to formalize a process in order to extract model instances from existing load intensity traces?). The Simple Model Instance Extraction Process and the Periodic Model Instance Extraction Process extract DLIM instances, with different *Trend Parts*, whereas the high-level Model Instance Extraction Process extracts an hl-DLIM instance.

The differences between the three processes were pointed out, as well as their degree of automation.

7. Evaluation

The Descartes Load Intensity Model as well as the high-level Descartes Load Intensity Model are evaluated on the basis of existing real world load intensities, based on 9 existing Web and Cloud server traces. The model extraction methods (see Chapter 6) are applied to these traces in order to extract DLIM and hl-DLIM instances. The arrival rate variation within these instances are compared to one another using the difference metrics described in Section 7.1.2.2. I also compare the model extraction methods' accuracy and run-time with BFAST time-series decomposition.

The applicability and usability of the model for custom load variation creation is evaluated separately based on a case study. Computer scientists from industry and academia (some of them already using results of this thesis' work for their purposes) answered a questionnaire about the model's applicability and the LIMBO's usability for their purposes.

Finally, I perform a short evaluation on whether future work on DLIM for load intensity forecasting is warranted.

7.1. Evaluating Model Accuracy and the Model Extraction Process

I evaluate the presented model extraction methods based on 9 different real-world Web and Cloud server traces covering between two weeks and seven months.

The first batch of traces was retrieved from The Internet Traffic Archive[1]. They were parsed to arrival rate traces of a quarter-hourly resolution (meaning 96 arrival rate samples per day).

- **ClarkNet-HTTP**: HTTP requests to the ClarkNet WWW server between August 28, 1995 at 00:00 and September 10th, 1995 at 23:59. ClarkNet is an Internet access provider in the Metro Baltimore-Washington DC area.
- **NASA-HTTP**: HTTP requests to the NASA Kennedy Space Center WWW server between July 1st, 1995 at 00:00 and August 31st, 1995 at 23:59.
- **Saskatchewan-HTTP**: HTTP requests to the University of Saskatchewan's WWW server between June 1st, 1995 at 00:00 and December 31st, 1995 at 23:59.

[1]Internet Traffic Archive: `http://ita.ee.lbl.gov`

- **WorldCup98**: Web requests recorded at servers for the 1998 World Cup between April 30th, 1998 at 22:00 UTC and June 8th, 1998 at 22:00 UTC.

The second batch of traces was retrieved from the Wikipedia page view statistics[2].They were parsed from the *projectcount* dumps, which already feature hourly arrival rates. All traces are from December 2013, with the exception of the English Wikipedia trace, which is from November 2013. The English December 2013 trace features a major irregularity during the 4th day, which I attribute to a measurement or parsing error.

- **de.wikipedia.org**: Requests to all German Wikipedia projects during December 2013.
- **fr.wikipedia.org**: Requests to all French Wikipedia projects during December 2013.
- **ru.wikipedia.org**: Requests to all Russian Wikipedia projects during December 2013.
- **wikipedia.org**: Requests to all English Wikipedia projects during November 2013.

The Wikipedia traces feature an access behavior over different time zones. The German Wikipedia projects are mostly accessed from a single time zone (CET), whereas the English Wikipedia is accessed from all over the world. Evaluating the Wikipedia traces thus also helps to asses how well the DLIM extraction processes deal with local vs. global access patterns.

The final trace is a trace of transactions on an IBM z-Series Mainframe [Vau13] from February 2011. It already features arrival rates using a quarter-hourly resolution.

For each trace DLIM instances are extracted using S-MIEP. Different *Trend* lengths are compared for the resulting model instances. Best results are expected at *Trend* length of one *Seasonal* period, whereas lower accuracy is expected at the longest evaluated *Trend* length of three *Seasonal* periods. I also evaluate the effect of noise reduction and extraction, by comparing an extracted model instance that was not extracted using noise reduction and extraction with an instance that features extracted noise and an instance that was extracted using noise reduction (the Gaussian filter, see 7.1.1.1).

For traces with a duration greater than one month, I also apply P-MIEP. P-MIEP is configured to extract weeks as the periodic *Trend* list with two *Trend* segments of the length 3 and 4. Additionally it extracts a bi-weekly period with a *Trend* list using two *Trend* segments of the length 7. Finally, it extracts a monthly (4-weekly) period with a *Trend* list using two *Trend* segments of the length 14. I also evaluate the impact of noise reduction and extraction by comparing the three different extracted model instances.

I also extract hl-DLIM parameters from the traces (see Section 6.5) using hl-MIEP and discuss the impact of noise reduction and extraction as part of the evaluation.

The shape of the interpolating functions is always selected as the DLIM *SinTrend*, meaning that sin flanks are always used for the interpolation between arrival rate peaks and lows. I choose *SinTrend* because it fits closest to the original trace in the majority of cases. An advance interpolation function selection will have to be part of future work.

For the same reasons as the *SinTrend* selection for *SeasonalShape* and *TrendShape*, *ExponentialIncreaseAndDecline* is always selected for *Burst* modeling. *Trends* are always selected to be multiplicative since this way they have a lower impact on arrival rate lows and a relatively high impact on arrival rate peaks (contrary to additive *Trends*, which have a constant impact on both). I do this, since arrival rate lows seem to vary less than arrival rate peaks according to my observations.

[2]Wikipedia project-counts: `http://dumps.wikimedia.org/other/pagecounts-raw/`

7.1.1. Trace Requirements

Considering that the model extraction methods and hl-DLIM are specifically designed for load intensity variations with a seasonal base part and overarching trends, which are caused by human motivated request arrival rate behavior, the traces used for the extraction process and hl-DLIM evaluation should be selected to mirror these considerations.

The requirements for traces to be used as part of the evaluation process are therefore:

- Requests within the traces must originate from human users.

  This requirement excludes traces of requests within any fully automated system, such as scientific computing grids, or similar systems.

- The traces must have a duration of at least several days.

  Since most human based activities have an inherent seasonal period based on the daily cycle it stands to reason that traces with a duration of multiple days contain some sort of repeating seasonal pattern.

- The traces should have minimal noise.

  Noise adds an additional level of difficulty to the model extraction process. For more details on this requirement see Section 7.1.1.1.

7.1.1.1. Noise Reduction

Noise adds an additional difficulty to the model extraction. It must first be eliminated in order to make the seasonal and trend parts more visible.

Since automatic model extraction was not part of the original scope of this work, noise reduction for automatic model extraction was only done superficially. A Gaussian filter attempts to limit the impact of noise by removing high frequencies from the trace. Unfortunately, this may lead to decreased model accuracy of the extracted model instance.

I attempt to limit the impact of existing noise in the traces by clever use of the sampling frequency. By allowing a seasonal period to only contain between 20 and 100 sampled arrival rates, noise is expected to have less of an impact on the min-max search employed as part of the model extraction process.

I do evaluate the impact of the Gaussian filter on overall process accuracy. Future work should, however, use more advanced techniques in order to remove noise. This could lead to a considerable improvement for any automated model extraction.

7.1.2. Evaluation Metrics

I calculate the difference between the original trace and the extracted model instance using LIMBO's difference calculator (see Section 5.7.1). The difference calculator returns a list of absolute differences between each arrival rate defined in the trace and the arrival rate produced by the model at its respective time-stamp. It also returns the overall relative absolute mean and median differences. It does so by calculating the relative absolute arrival rate difference at each sampled point in time:

$$relative\Delta = |\tfrac{arrivalRate_{read} - arrivalRate_{model}}{arrivalRate_{read}}|$$

The mean and median relative absolute differences are then derived from the list of all relative absolute differences.

Additionally the difference calculator features a Dynamic Time Warping (DTW) [Mül07] based difference metric.

Typical statistical test, such as the F-Test, (paired) T-Test, or Chi^2-Test were not performed on the extracted model instances, as they ignore the time-component when evaluating the distribution of measured arrival rates. Since the time-dependent change of the arrival rate variations is an integral part of DLIM and the extraction methods, these statistical test are unfit for the comparison of the arrival rate distributions.

7.1.2.1. Motivating the Dynamic Time Warping-based Difference Metric

The median and mean arrival rate difference features a major problem: It does not take the time-stamp of the arrival rates into account. The two arrival rate variation profiles in Fig. 7.1 illustrate this. These two arrival rate variation profiles are clearly identical, yet one lags 5 seconds behind. Simply calculating the arrival rate difference does not take this into account (mean difference being at 37.5%, median difference being at 10%).

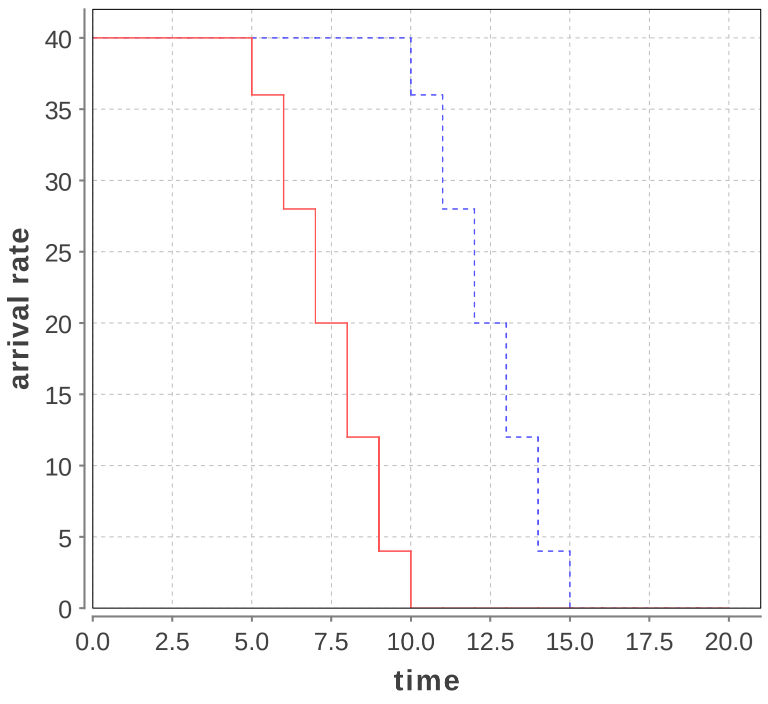


Figure 7.1.: Two identical, yet time-wise offset arrival rate variation profiles.

This problem can be solved by applying *Dynamic Time Warping (DTW)* [Mül07] to the two-dimensional data space consisting of both the time-stamps and their respective arrival rates. DTW can then find the closest data point in this two dimensional space and calculate the distance between these two points (using the euclidean distance).

For the example arrival rate variations in Fig. 7.1, the DTW based curve difference is thus at 0.06875.

7.1.2.2. Difference Metric based on DTW

The difference calculator provides an additional difference metric based on the FastDTW [SC07] implementation of the *Dynamic Time Warping (DTW)* algorithm. It calculates the distance of the trace arrival rate time series and the model's arrival rates at the times specified within the trace. The euclidean distance is used to calculate the distance of the two-dimensional data points consisting of the arrival rates and their respective time-stamps (the time-stamps have to be taken into account in order to detect and penalize identical yet time-wise offset arrival rate functions). The resulting DTW distance is then normalized by division through the amount of data points (arrival rate samples) as well as division through the maximum arrival rate in the trace.

This DTW-based difference metric is thus defined as 1 for the difference between a constant function (defined by the trace's maximum value) and the constant zero-function, and 0 for the two identical arrival rate functions. Each point in time, at which the data points in the DLIM instance and trace are closer to one another, diminishes the overall DTW-distance.

The DTW-based difference metric allows easy comparison of different model instances, which are based on the same arrival rate trace, as the metric is always normalized to the trace's maximum arrival rate value.

7.1.3. ClarkNet-HTTP

The ClarkNet-HTTP trace[3] features two weeks of HTTP requests to the ClarkNet WWW server between August 28, 1995 at 00:00 and September 10th, 1995 at 23:59. ClarkNet is an Internet access provider in the Metro Baltimore-Washington DC area. I parsed the trace to an arrival rate trace with a quarter-hourly resolution.

As a result the *Seasonal Part* is extracted from 96 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

Both extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

P-MIEP was not performed on this trace, since it only features two weeks of requests and is thus too short repeating weekly, bi-weekly, and monthly patterns.

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
S-MIEP, Trend length 1, noise extracted	21.195	26.0	0.051992
S-MIEP, Trend length 1, noise reduced	17.509	20.475	0.046219
S-MIEP, Trend length 1, noise ignored	**12.409**	15.753	0.037891
S-MIEP, Trend length 2, noise ignored	14.734	20.343	0.044982
S-MIEP, Trend length 3, noise ignored	14.919	19.542	0.043292
hl-MIEP, Trend length 1, noise extracted	20.105	22.479	0.043371
hl-MIEP, Trend length 1, noise reduced	19.361	21.477	0.042963
hl-MIEP, Trend length 1, noise ignored	72.924	66.024	0.269728

Table 7.1.: ClarkNet-HTTP model extraction accuracy.

The ClarkNet extraction results in Table 7.1 and Figures 7.2 and 7.3 show that S-MIEP offers by far the best results, especially with a *Trend* length of 1. Noise elimination does not seem to help for this particular trace during the DLIM extraction. The result is also not optimal when extracting noise, since the randomly generated noise does not produce the exact same results as the original noise, thus increasing the difference between model and original trace arrival rates.

[3]ClarkNet-HTTP: `http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html`

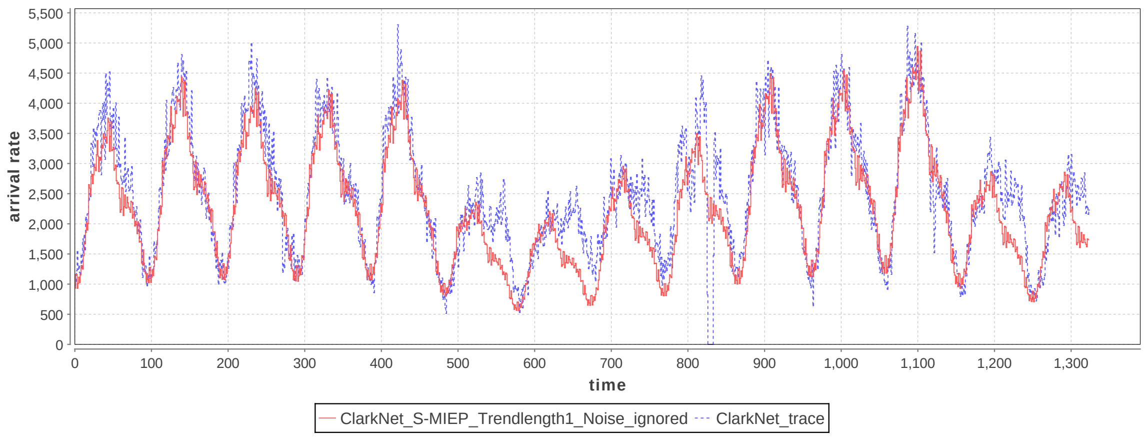


Figure 7.2.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

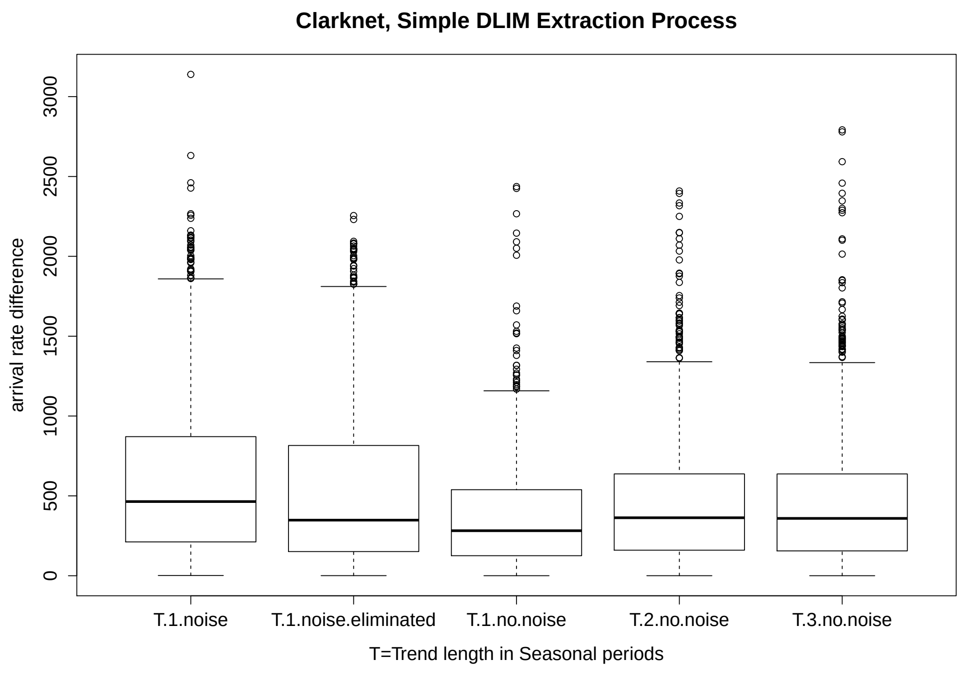


Figure 7.3.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

Hl-MIEP shows another picture entirely. Considering that hl-DLIM only offers a small number of pre-defined parameters, the extracted hl-DLIM instances are surprisingly close to the original model. Contrary to what was observed in the DLIM extraction, the hl-DLIM extraction heavily relies on the noise reduction. Hl-MIEP accuracy is significantly improved once noise is filtered and not ignored. This can easily be attributed to the linear interpolation between extracted peaks. Since hl-DLIM interpolates between the highest and lowest peak (thus only extracting two peaks), the non-filtered trace offers a great number of noisy peaks with minimal impact on the overall arrival rate. The filtered version however only offers a few remaining peaks, which have a far greater impact on the overall arrival rate. Applying noise reduction forces hl-MIEP to only consider the remaining peaks with greater impact and not accidentally choosing an outlier for its peaks.

7.1.4. NASA-HTTP

The NASA-HTTP trace[4] features two months of HTTP requests to the NASA Kennedy Space Center WWW server between July 1st, 1995 at 00:00 and August 31st, 1995 at 23:59. I parsed the trace to an arrival rate trace with a quarter-hourly resolution.

As a result the *Seasonal Part* is extracted from 96 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

P-MIEP is configured to extract weeks as the periodic *Trend* list with two *Trend* segments of the length 3 and 4. Additionally it extracts a bi-weekly period with a *Trend* list using two *Trend* segments of the length 7. Finally, it extracts a monthly (4-weekly) period with a *Trend* list using two *Trend* segments of the length 14.

All extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
P-MIEP, noise extracted	32.223	87.619	0.026951
P-MIEP, noise reduced	28.944	85.013	0.024968
P-MIEP, noise ignored	23.633	91.185	0.028573
S-MIEP, Trend length 1, noise extracted	26.446	77.651	0.022533
S-MIEP, Trend length 1, noise reduced	23.56	73.865	0.020487
S-MIEP, Trend length 1, noise ignored	**18.812**	69.113	0.018773
S-MIEP, Trend length 2, noise ignored	20.8	77.354	0.02209
S-MIEP, Trend length 3, noise ignored	27.577	83.306	0.030999
hl-MIEP, Trend length 1, noise extracted	26.541	76.31	0.020706
hl-MIEP, Trend length 1, noise reduced	24.539	73.531	0.021203
hl-MIEP, Trend length 1, noise ignored	55.575	64.494	0.05495

Table 7.2.: NASA-HTTP model extraction accuracy.

The NASA-HTTP extraction results in Table 7.2 and Figures 7.4 and 7.5, while not performing quite as well as in the ClarkNet-HTTP extraction, enforce several of the conclusions from the ClarkNet trace evaluation. S-MIEP still provides the best results, especially with *Trend* length 1, and noise reduction seems to be useful for hl-MIEP.

[4]NASA-HTTP: `http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html`

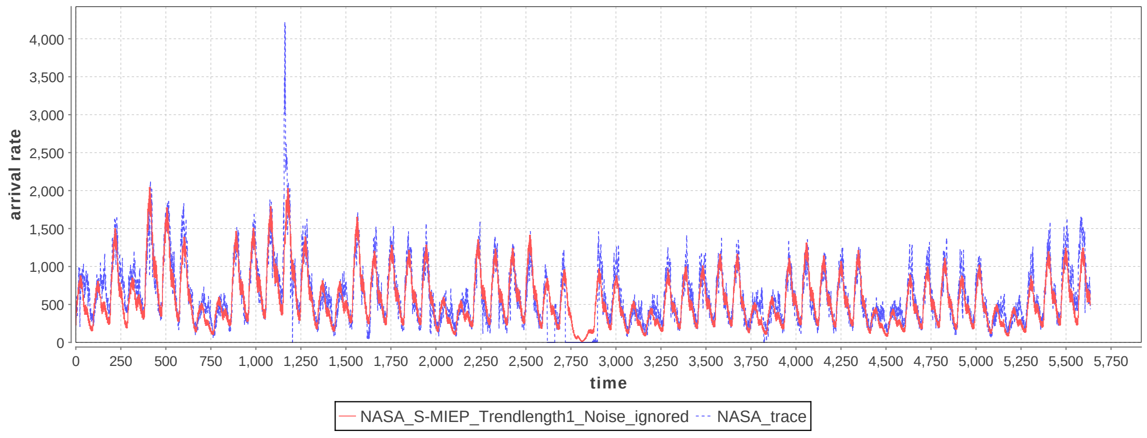


Figure 7.4.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

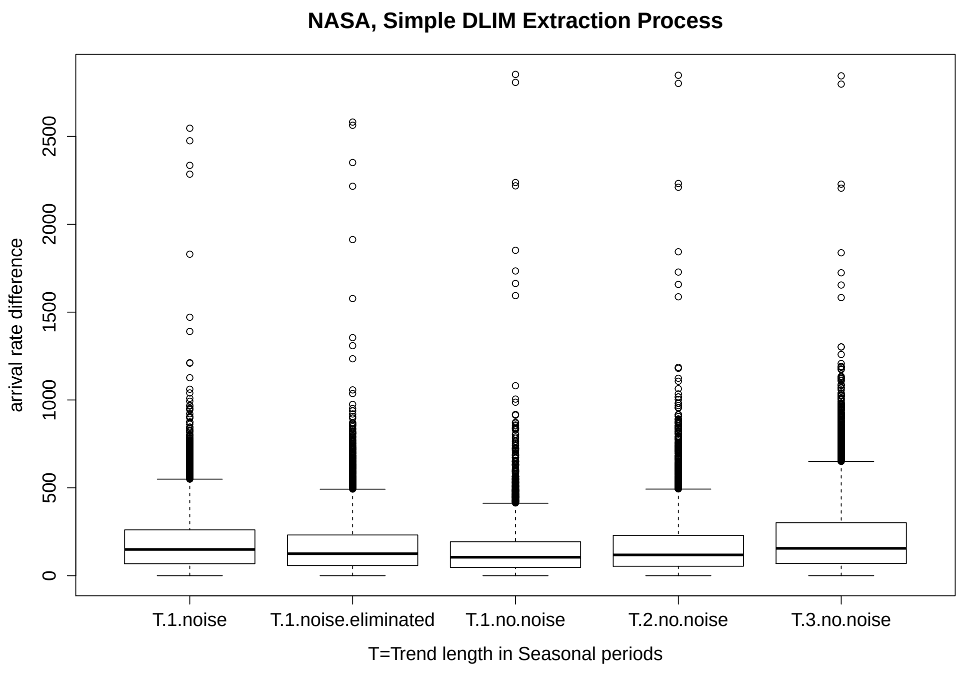


Figure 7.5.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

The discrepancies between extracted model instance and original trace seem to have three major causes:

- Burst detection fails to notice a major burst at around time-stamp 1200.
- The server was shut down for maintenance between time-stamps 2700 and 2900. The extraction processes do not have a contingency for this case.
- **Deviating Seasonal Patters**: This is a major cause for all deviations in all evaluated traces. The extraction processes all assume a single, identically repeating *Seasonal Part*. Depending on the trace, this assumption may be valid to a different extent. In this case, the extracted Seasonal pattern is able to approximate most days in the trace, but a number of significant deviations do occur. Manual modeling in the DLIM editor can circumvent this problem, hl-DLIM and the automated extractors however are not able to do this.

P-MIEP performs well when being compared to the other two extraction processes. This is surprising in that the periodic process assumes that all trends repeat. In this case, this assumption seems to be relatively accurate. As a result, this trace might be well suited for workload forecasting.

7.1.5. Saskatchewan-HTTP

The Saskatchewan-HTTP trace[5] features seven months of HTTP requests to the University of Saskatchewan's WWW server between June 1st, 1995 at 00:00 and December 31st, 1995 at 23:59. I parsed the trace to an arrival rate trace with a quarter-hourly resolution.

As a result the *Seasonal Part* is extracted from 96 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

P-MIEP is configured to extract weeks as the periodic *Trend* list with two *Trend* segments of the length 3 and 4. Additionally it extracts a bi-weekly period with a *Trend* list using 2 *Trend* segments of the length 7. Finally, it extracts a monthly (4-weekly) period with a *Trend* list using 2 *Trend* segments of the length 14.

All extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

The Saskatchewan-HTTP extraction results in Table 7.3 and Figures 7.6 and 7.7 show a different picture than the previous trace extraction results. First off, in this case noise reduction also improves S-MIEP results. Secondly the extraction results can not reach the same accuracy as in the ClarkNet and NASA extraction.

The major explanation for the relatively poor results for this trace is once more the *Seasonal* pattern deviation. Since the Saskatchewan-HTTP trace extends over 7 months, the *Seasonal* patterns have a lot of room for deviation. The model extractors fail to capture this. This leads to an additional error in the *Trend* calibration. Since the trends are supposed to be calibrated, so that each greatest *Seasonal* peak matches the trace's nearest local arrival rate maximum. Since the greatest extracted *Seasonal* peak's time of day does not match the trace's greatest *Seasonal* peak's time of day, the calibration takes place at the wrong time-stamp. This explains why a majority of extracted days have a lower peak than their counterparts in the original trace.

[5]Saskatchewan-HTTP: `http://ita.ee.lbl.gov/html/contrib/Sask-HTTP.html`

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
P-MIEP, noise extracted	43.293	78.011	0.035426
P-MIEP, noise reduced	35.831	62.666	0.035263
P-MIEP, noise ignored	35.663	60.937	0.033896
S-MIEP, Trend length 1, noise extracted	35.551	68.764	0.029433
S-MIEP, Trend length 1, noise reduced	**26.492**	49.736	0.030491
S-MIEP, Trend length 1, noise ignored	29.171	50.37	0.026332
S-MIEP, Trend length 2, noise ignored	30.273	53.303	0.027575
S-MIEP, Trend length 3, noise ignored	32.085	58.495	0.029935
hl-MIEP, Trend length 1, noise extracted	37.942	60.15	0.031448
hl-MIEP, Trend length 1, noise reduced	33.24	52.568	0.034878
hl-MIEP, Trend length 1, noise ignored	80.792	75.635	0.100943

Table 7.3.: Saskatchewan-HTTP model extraction accuracy.

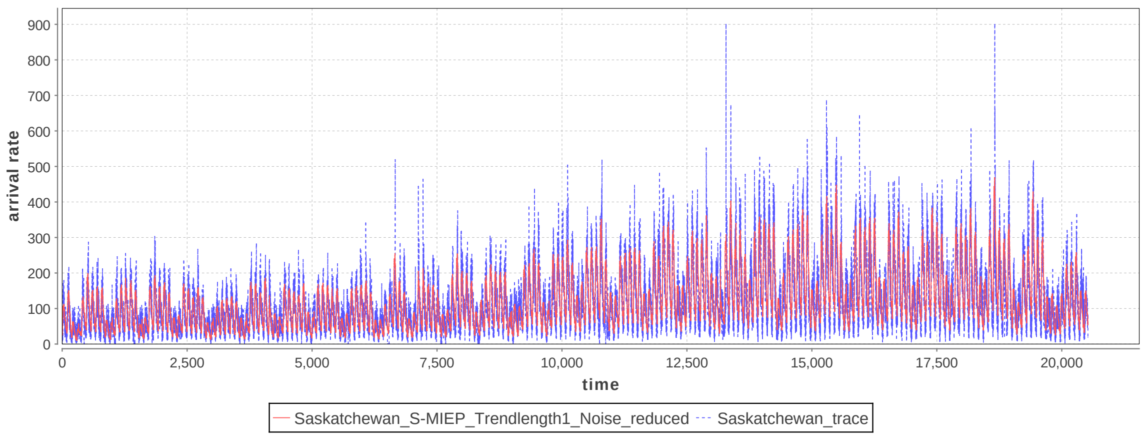


Figure 7.6.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and with noise having been eliminated (red).

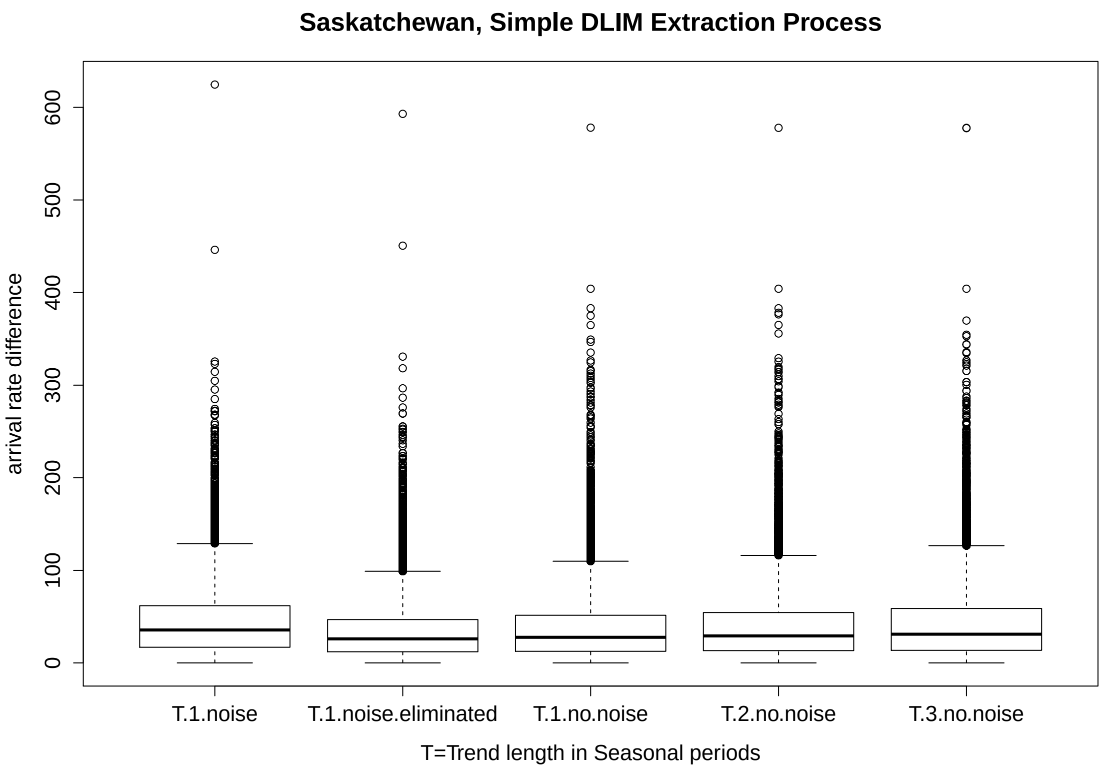


Figure 7.7.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

The major deviation from the trace's *Seasonal* patterns also explains why S-MIEP performs better using noise elimination for the Saskatchewan-HTTP extraction. Noise reduction helps to mitigate the effect of seasonal pattern changes over time, thus reducing the effect of the *Seasonal* pattern deviation.

7.1.6. WorldCup98

The WorldCup98 traces[6] features Web requests recorded at servers for the 1998 World Cup. I only used the requests between April 30th, 1998 at 22:00 UTC and June 8th, 1998 at 22:00 UTC, considering the trace is extremely large. I parsed the trace to an arrival rate trace with a quarter-hourly resolution.

As a result the *Seasonal Part* is extracted from 96 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

P-MIEP is configured to extract weeks as the periodic *Trend* list with 2 *Trend* segments of the length 3 and 4. Additionally it extracts a bi-weekly period with a *Trend* list using 2 *Trend* segments of the length 7. Finally, it extracts a monthly (4-weekly) period with a *Trend* list using 2 *Trend* segments of the length 14.

All extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

[6]WorldCup98: `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`

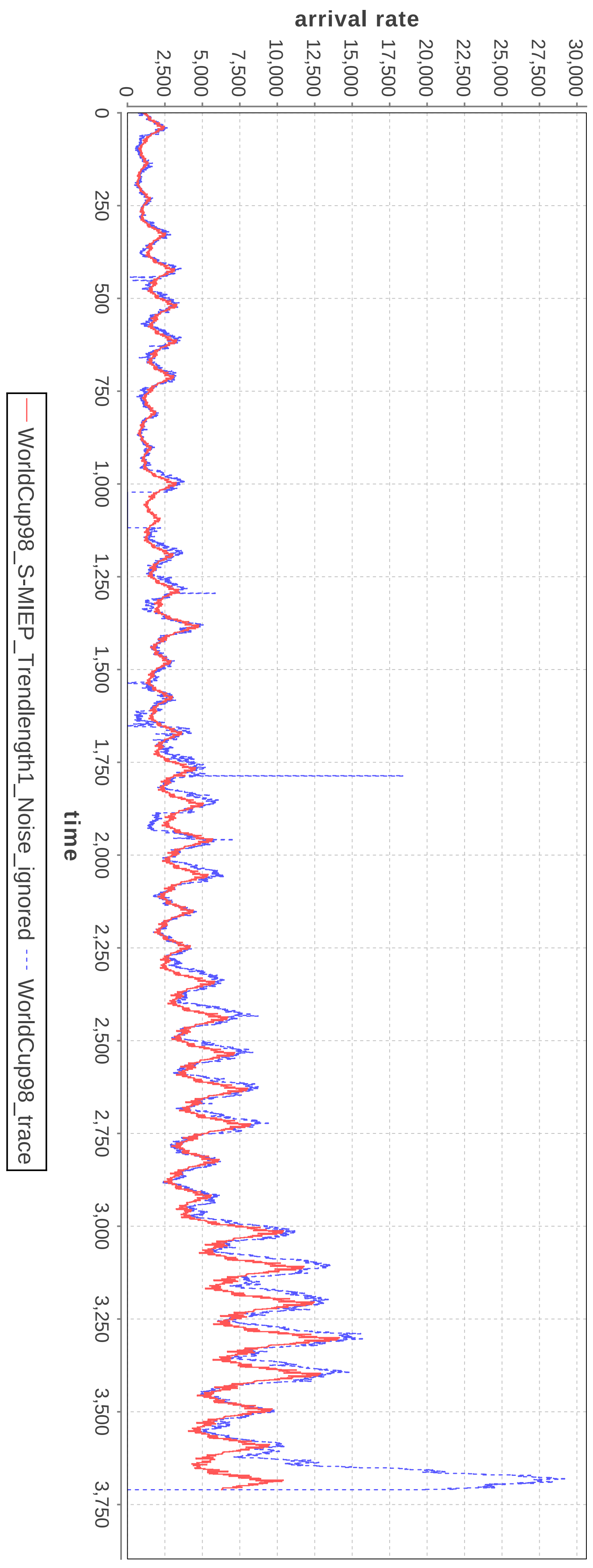


Figure 7.8.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
P-MIEP, noise extracted	52.304	63.847	0.047354
P-MIEP, noise reduced	53.316	61.522	0.045723
P-MIEP, noise ignored	53.495	56.625	0.069748
S-MIEP, Trend length 1, noise extracted	19.735	32.485	0.017751
S-MIEP, Trend length 1, noise reduced	16.882	27.575	0.014515
S-MIEP, Trend length 1, noise ignored	**12.979**	22.157	0.018861
S-MIEP, Trend length 2, noise ignored	15.691	25.458	0.020503
S-MIEP, Trend length 3, noise ignored	19.161	28.886	0.019215
hl-MIEP, Trend length 1, noise extracted	16.093	26.562	0.017208
hl-MIEP, Trend length 1, noise reduced	15.66	25.995	0.017179
hl-MIEP, Trend length 1, noise ignored	43.957	43.129	0.061156

Table 7.4.: WorldCup98 model extraction accuracy.

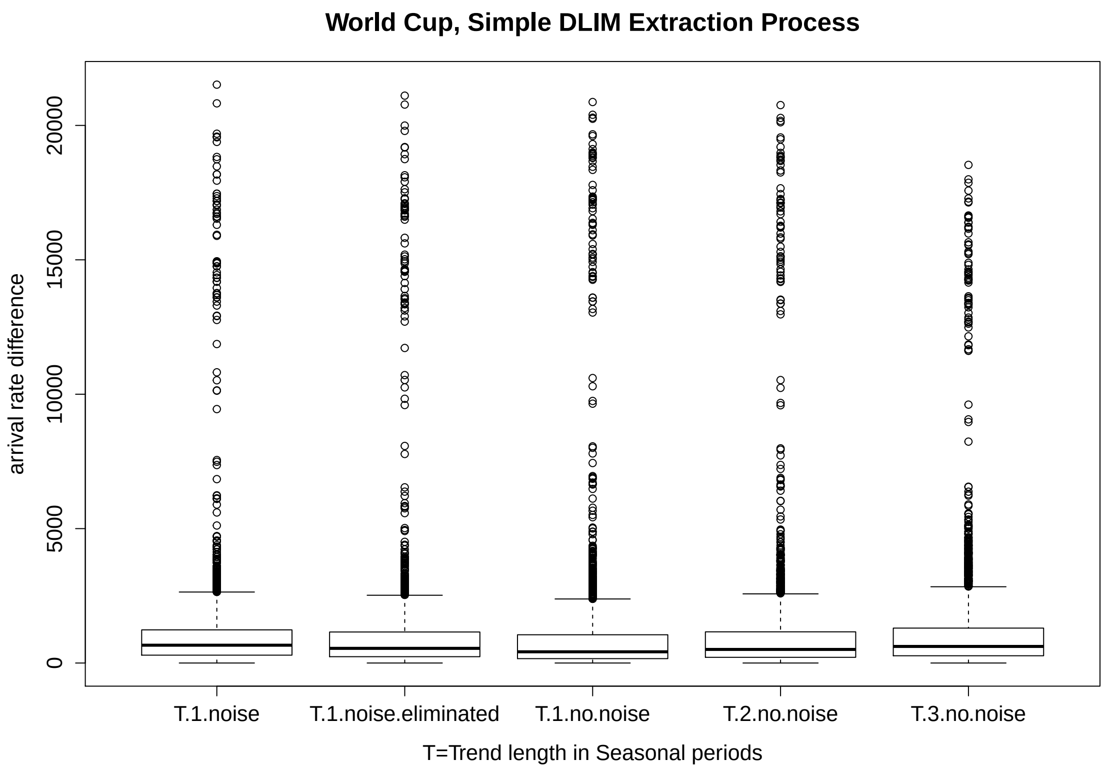


Figure 7.9.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

The WorldCup98 extraction results in Table 7.4 and Figures 7.8 and 7.9 follow the patterns of the previous trace extraction results. Noise elimination is useful for hl-MIEP, shorter trends produce better results, with S-MIEP producing the best results.

Notable however is that P-MIEP performs worst of all considered traces, especially since the other two processes fare relatively well. The obvious cause of this is the observation that the WorldCup98 trace does not feature repeating trends and it only features increasing trends. The S-MIEP and hl-MIEP can handle this easily, whereas P-MIEP cannot.

The major error cause is once again the *Seasonal* pattern deviation. Additionally, the last day is not interpolated correctly. The trend interpolation ignores it, as it is not a complete day within the trace (the trace cuts off about 18 hours into the last day), and the trace interpolation ignores incomplete *Seasonal* periods at the end of the trace.

7.1.7. German Wikipedia

This trace contains the requests to all German Wikipedia projects during December 2013[6]. It already contained arrival rates with an hourly resolution.

As a result the *Seasonal Part* is extracted from 24 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

Both extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

P-MIEP was not performed on this trace, since it only features one month of requests and is thus too short repeating monthly patterns.

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
S-MIEP, Trend length 1, noise extracted	11.215	16.623	0.027493
S-MIEP, Trend length 1, noise reduced	10.511	16.013	0.026259
S-MIEP, Trend length 1, noise ignored	**8.538**	16.55	0.027512
S-MIEP, Trend length 2, noise ignored	9.956	17.659	0.031427
S-MIEP, Trend length 3, noise ignored	11.771	17.884	0.03574
hl-MIEP, Trend length 1, noise extracted	11.898	19.26	0.027019
hl-MIEP, Trend length 1, noise reduced	11.393	18.991	0.027193
hl-MIEP, Trend length 1, noise ignored	13.126	23.392	0.030153

Table 7.5.: de.wikipedia.org model extraction accuracy.

The de.wikipedia.org extraction results in Table 7.5 and Figures 7.10 and 7.11 confirm many of the observations made with the Internet Traffic Archive traces. Noise extraction is most useful for hl-MIEP, *Trend* length 1 as part of S-MIEP performs best. The overall accuracy, however, is significantly better than for the Internet Traffic Archive traces since the *Seasonal* pattern deviation, while still relevant, shows less impact than before.

[6]Wikipedia project-counts, December 2013: `http://dumps.wikimedia.org/other/pagecounts-raw/2013/2013-12/`

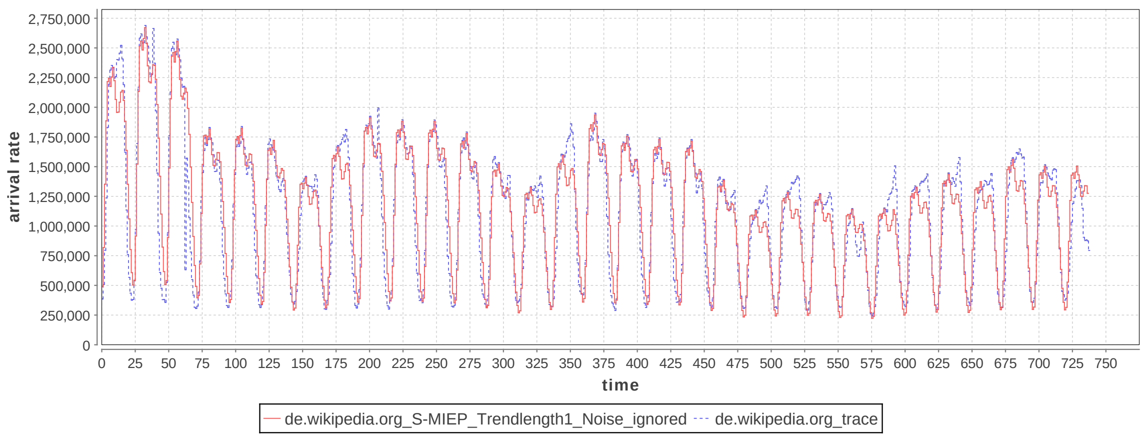


Figure 7.10.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

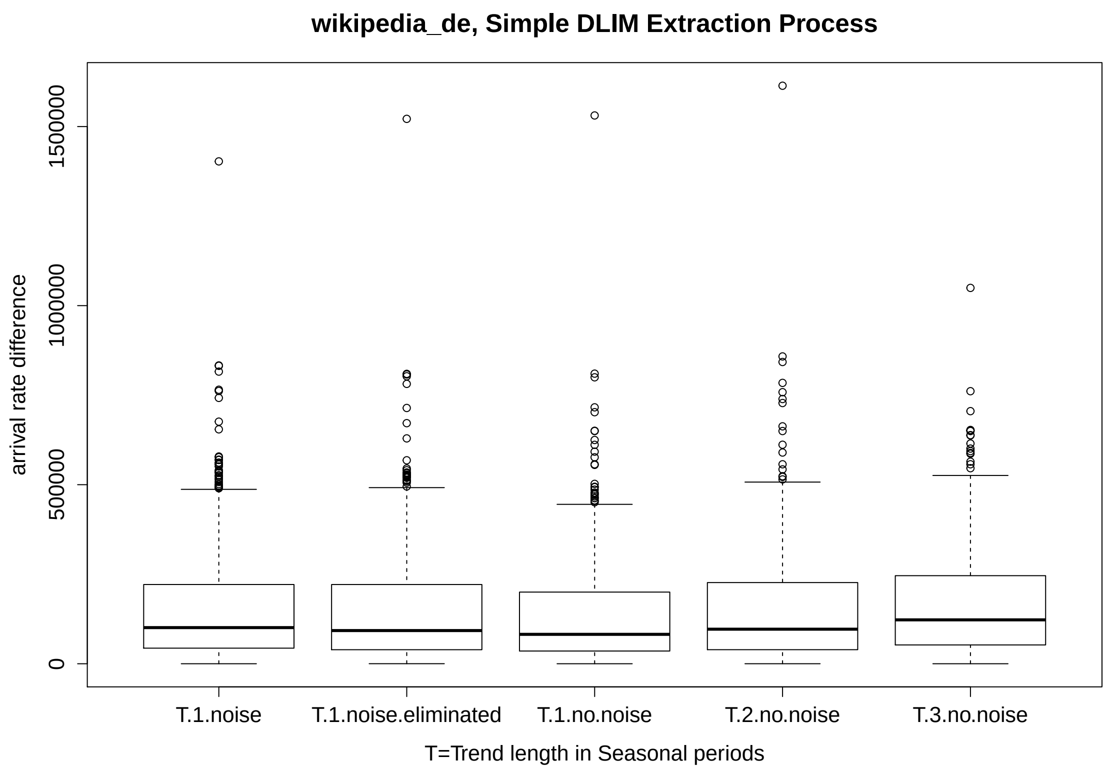


Figure 7.11.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

7.1.8. French Wikipedia

This trace contains the requests to all French Wikipedia projects during December 2013[6]. It already contained arrival rates with an hourly resolution.

As a result the *Seasonal Part* is extracted from 24 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

Both extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

P-MIEP was not performed on this trace, since it only features one month of requests and is thus too short repeating monthly patterns.

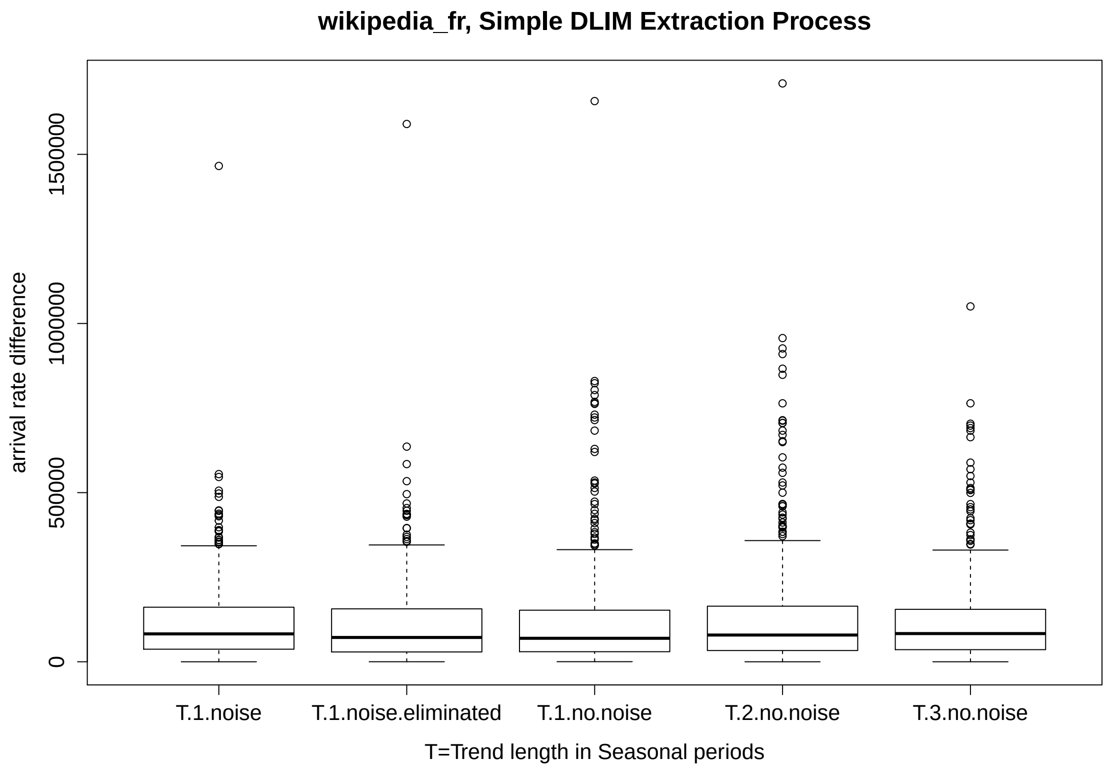


Figure 7.12.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

The fr.wikipedia.org extraction results in Table 7.6 and Figures 7.13 and 7.12 confirm previous observations. Noise extraction is most useful for hl-MIEP, *Trend* length 1 as part of S-MIEP performs best. The overall accuracy is similar to the de.wikipedia.org trace extraction, since the *Seasonal* pattern deviation, while still relevant, has similarly little impact.

[6]Wikipedia project-counts, December 2013: `http://dumps.wikimedia.org/other/pagecounts-raw/2013/2013-12/`

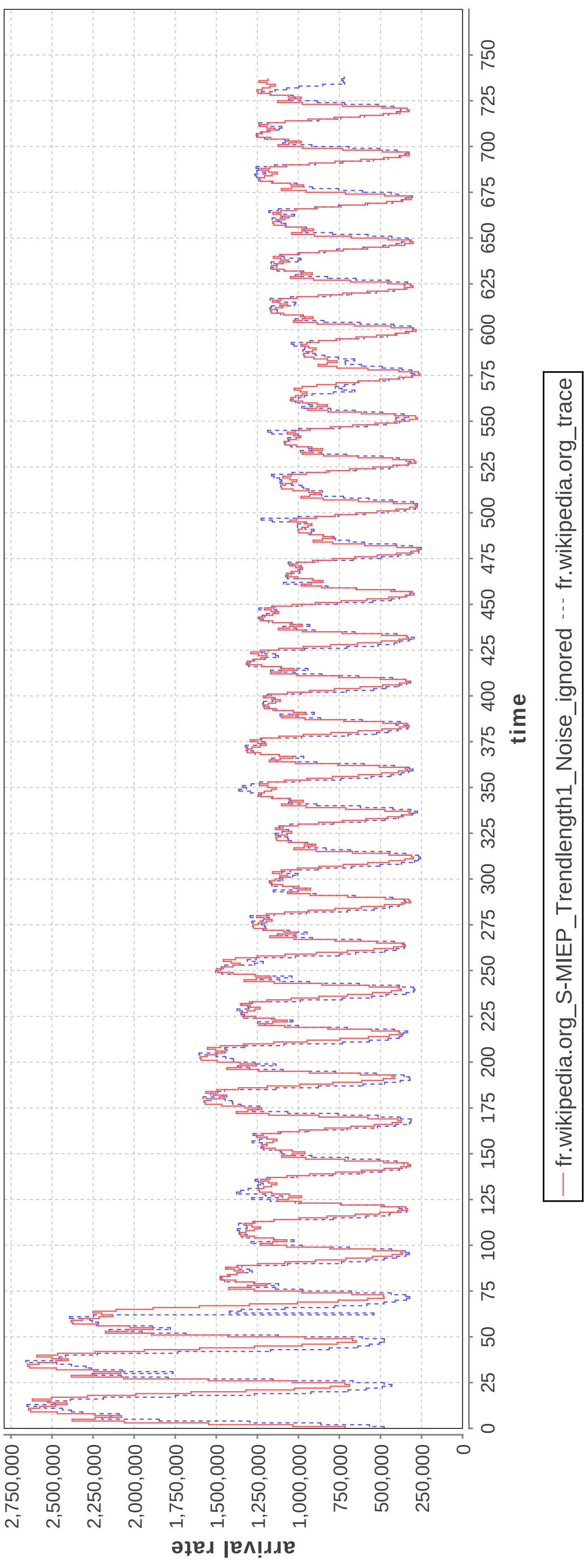


Figure 7.13.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

Extraction Parameters	relative median error (%)	relative mean error (%)	DTW curve difference
S-MIEP, Trend length 1, noise extracted	10.472	15.064	0.024289
S-MIEP, Trend length 1, noise reduced	8.566	13.984	0.022583
S-MIEP, Trend length 1, noise ignored	**7.6**	16.249	0.020954
S-MIEP, Trend length 2, noise ignored	8.973	17.393	0.02476
S-MIEP, Trend length 3, noise ignored	9.813	15.439	0.025621
hl-MIEP, Trend length 1, noise extracted	8.503	15.92	0.021719
hl-MIEP, Trend length 1, noise reduced	8.373	15.645	0.021834
hl-MIEP, Trend length 1, noise ignored	10.816	19.789	0.024962

Table 7.6.: fr.wikipedia.org model extraction accuracy.

7.1.9. Russian Wikipedia

This trace contains the requests to all Russian Wikipedia projects during December 2013[6]. It already contained arrival rates with an hourly resolution.

As a result the *Seasonal Part* is extracted from 24 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

Both extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

P-MIEP was not performed on this trace, since it only features one month of requests and is thus too short repeating monthly patterns.

Extraction Parameters	relative median error (%)	relative mean error (%)	DTW curve difference
S-MIEP, Trend length 1, noise extracted	9.964	12.99	0.019626
S-MIEP, Trend length 1, noise reduced	**9.912**	12.763	0.018085
S-MIEP, Trend length 1, noise ignored	11.251	13.768	0.024094
S-MIEP, Trend length 2, noise ignored	11.683	14.452	0.02651
S-MIEP, Trend length 3, noise ignored	11.42	14.249	0.028279
hl-MIEP, Trend length 1, noise extracted	12.392	16.239	0.020092
hl-MIEP, Trend length 1, noise reduced	12.496	16.256	0.020114
hl-MIEP, Trend length 1, noise ignored	13.31	16.829	0.024575

Table 7.7.: ru.wikipedia.org model extraction accuracy.

The ru.wikipedia.org extraction results in Table 7.7 and Figures 7.14 and 7.15 differ from the previous Wikipedia traces in that noise reduction also improves S-MIEP, while also being useful for hl-MIEP. The overall accuracy is similar to the previous Wikipedia trace extractions. For this single trace, however, the *Seasonal* patterns are shaped so the noise reduction lessens the impact of *Seasonal* pattern deviation.

[6]Wikipedia project-counts, December 2013: `http://dumps.wikimedia.org/other/pagecounts-raw/2013/2013-12/`

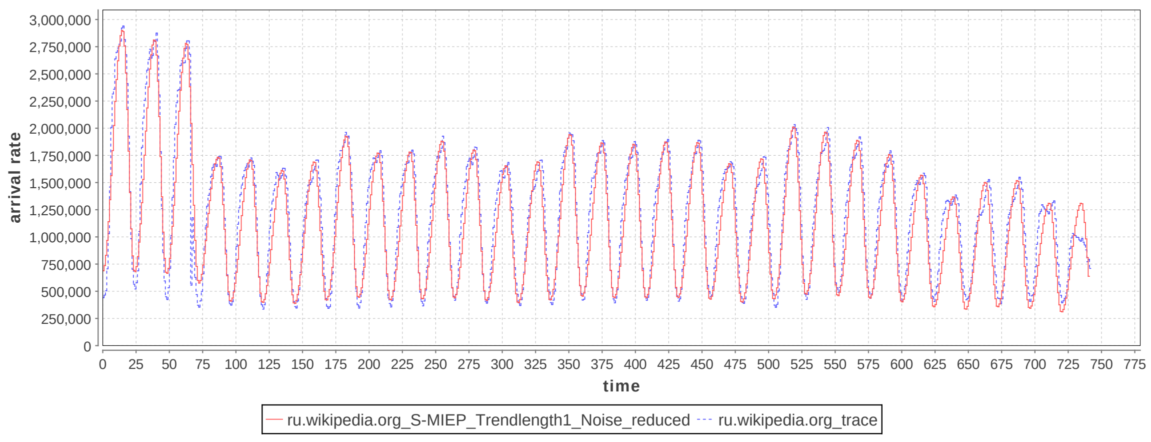


Figure 7.14.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and noise having been eliminated (red).

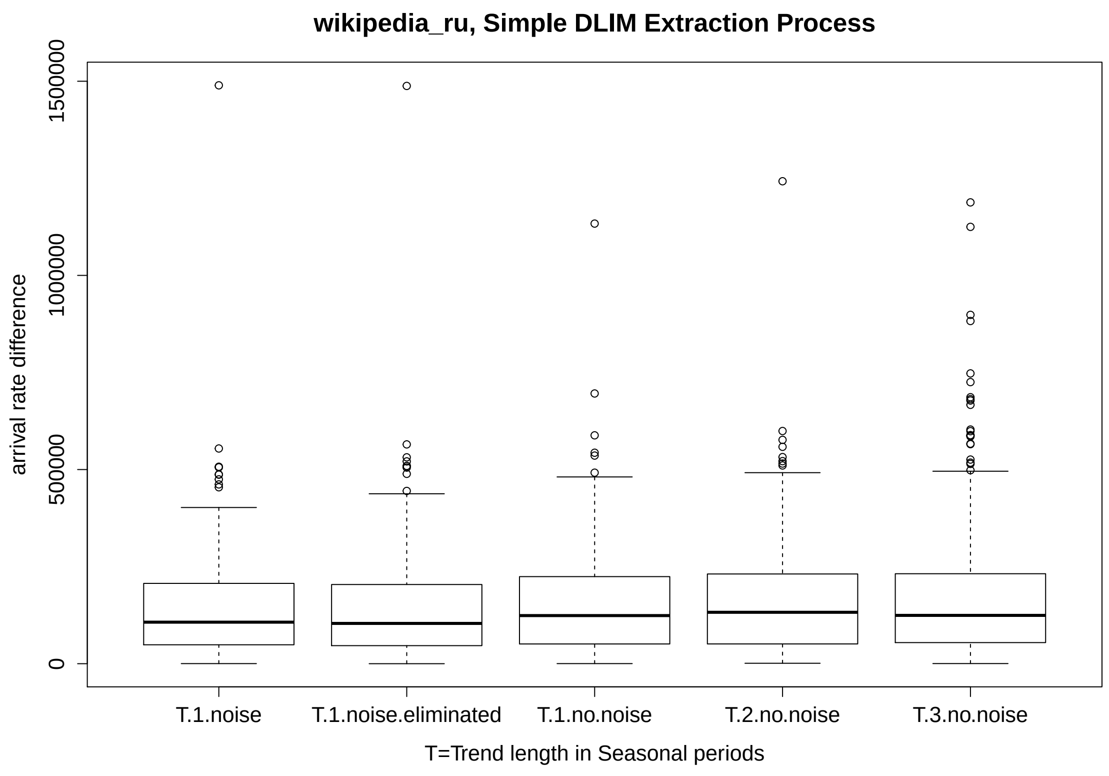


Figure 7.15.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

7.1.10. English Wikipedia

This trace contains the requests to all English Wikipedia projects during November 2013[7]. It already contained arrival rates with an hourly resolution.

As a result the *Seasonal Part* is extracted from 24 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

Both extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

P-MIEP was not performed on this trace, since it only features one month of requests and is thus too short repeating monthly patterns.

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
S-MIEP, Trend length 1, noise extracted	7.764	8.199	0.018225
S-MIEP, Trend length 1, noise reduced	7.838	8.074	0.017131
S-MIEP, Trend length 1, noise ignored	**4.855**	5.391	0.018488
S-MIEP, Trend length 2, noise ignored	5.27	6.346	0.022529
S-MIEP, Trend length 3, noise ignored	7.23	9.744	0.037233
hl-MIEP, Trend length 1, noise extracted	7.75	8.044	0.01765
hl-MIEP, Trend length 1, noise reduced	7.961	8.097	0.017208
hl-MIEP, Trend length 1, noise ignored	8.868	10.143	0.042072

Table 7.8.: wikipedia.org model extraction accuracy.

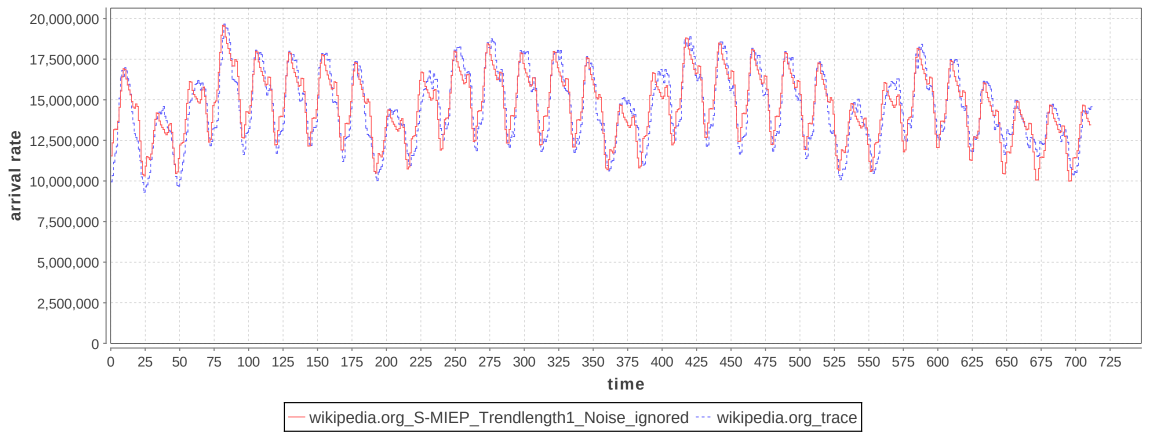


Figure 7.16.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

The wikipedia.org extraction results in Table 7.8 and Figures 7.16 and 7.17 are similar to the German and French Wikipedia extraction results. Noise extraction is most useful for hl-MIEP, *Trend* length 1 as part of S-MIEP performs best. The extraction results of

[7]Wikipedia project-counts, November 2013: `http://dumps.wikimedia.org/other/pagecounts-raw/2013/2013-11/`

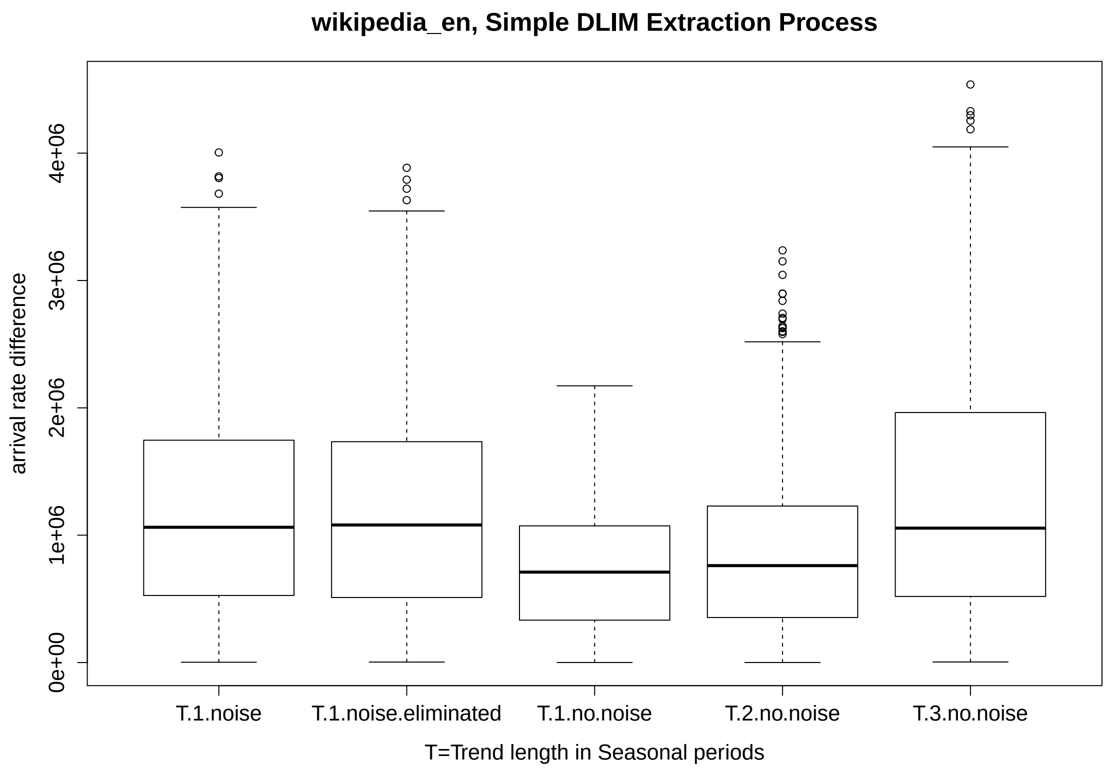


Figure 7.17.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

this trace, however, exhibit by far the best overall accuracy across all examined traces. The reason for this is the unusually high arrival rate base level. Since wikipedia.org is accessed globally at all times, the load intensity variations on top of the base level have little impact on the relative load variations in general. As a result, all modeling errors are also relatively small.

7.1.11. IBM z-Series Transactions

This trace contains the count of completed transactions on an IBM z196 Mainframe during February 2011 as used in [HHKA14]. The trace features the amount of CICS, IMS, and OPEN transactions with a quarter-hourly resolution.

As a result the *Seasonal Part* is extracted from 96 arrival rate samples.

For the *Trend Part* I extract different model instances with segment lengths between 1 and 3 *Seasonal* periods for S-MIEP. The hl-DLIM Extraction Process only uses a segment length of 1 *Seasonal* period for comparison with the DLIM extraction.

Both extraction processes are performed using noise reduction only, noise reduction and then extraction, and without any noise reduction and extraction for evaluation of the effect of noise on the extraction process.

P-MIEP was not performed on this trace, since it only features one month of requests and is thus too short repeating monthly patterns.

The IBM trace features an arrival rate variation that differs completely during weekdays and weekends. As a result *Seasonal* pattern deviation plays a major role. The single extracted seasonal pattern matches neither weekday nor weekend behavior and thus leads

Extraction Parameters	relative median error (%)	relative mean error (%)	DTW curve difference
S-MIEP, Trend length 1, noise extracted	95.351	431.165	0.033344
S-MIEP, Trend length 1, noise reduced	79.031	356.08	0.019388
S-MIEP, Trend length 1, noise ignored	74.368	124.603	0.040656
S-MIEP, Trend length 2, noise ignored	76.812	137.906	0.041428
S-MIEP, Trend length 3, noise ignored	**74.063**	220.94	0.038953
hl-MIEP, Trend length 1, noise extracted	91.05	254.992	0.034215
hl-MIEP, Trend length 1, noise reduced	85.613	229.08	0.029841
hl-MIEP, Trend length 1, noise ignored	84.972	487.071	0.079584

Table 7.9.: IBM z-Series Transactions model extraction accuracy.

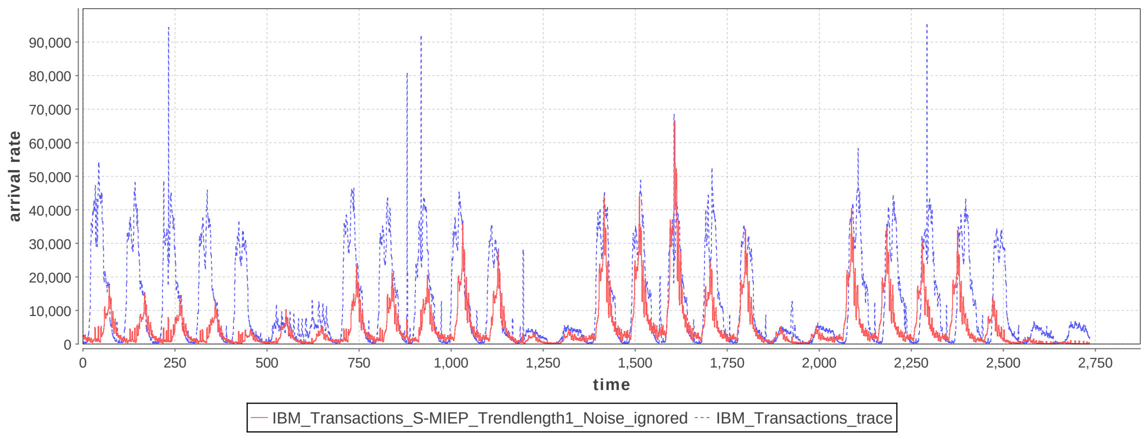


Figure 7.18.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

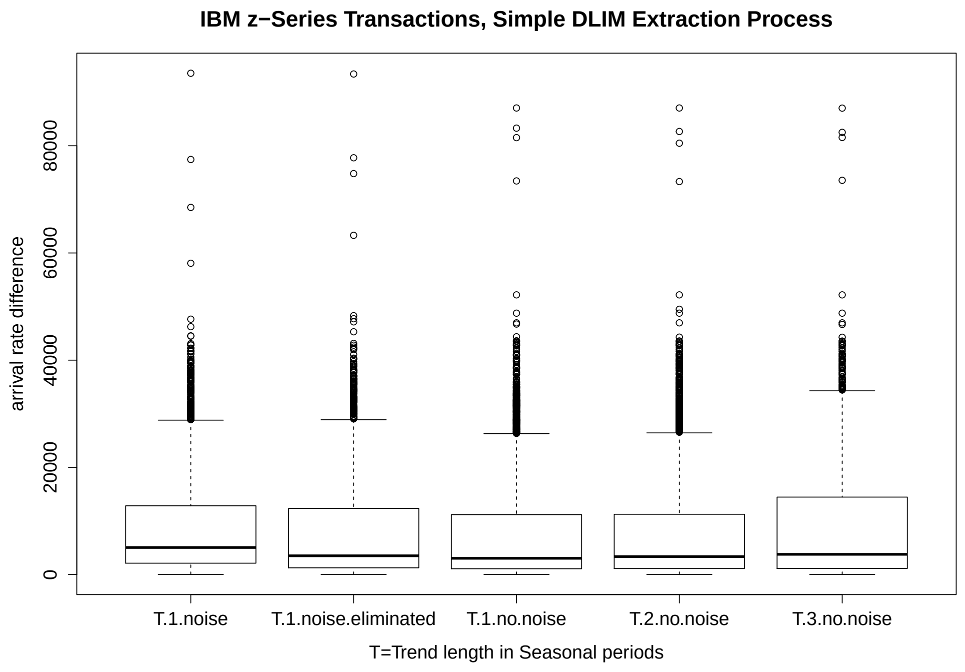


Figure 7.19.: Arrival rate accuracy of the different DLIM S-MIEP configurations.

a decrease in overall accuracy (see Table 7.9 and Figures 7.18 and 7.19). This trace shows a need for a future extraction process using multiple *Seasonal* patterns.

7.1.11.1. IBM z-Series Transactions during work days

To test whether *Seasonal* pattern deviation between work days and weekends is actually the main cause of the extraction error within the IBM z-Series trace, I extract a DLIM instance from the first 5 days within the trace. These days are all work days, as a result the problem of *Seasonal* pattern deviation should not have as much of an impact on the extraction.

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
S-MIEP, Trend length 1, noise extracted	89.828	336.09	0.033807
S-MIEP, Trend length 1, noise reduced	74.43	260.609	0.022686
S-MIEP, Trend length 1, noise ignored	**22.946**	101.132	0.019704

Table 7.10.: IBM z-Series Transactions model extraction accuracy (work days only).

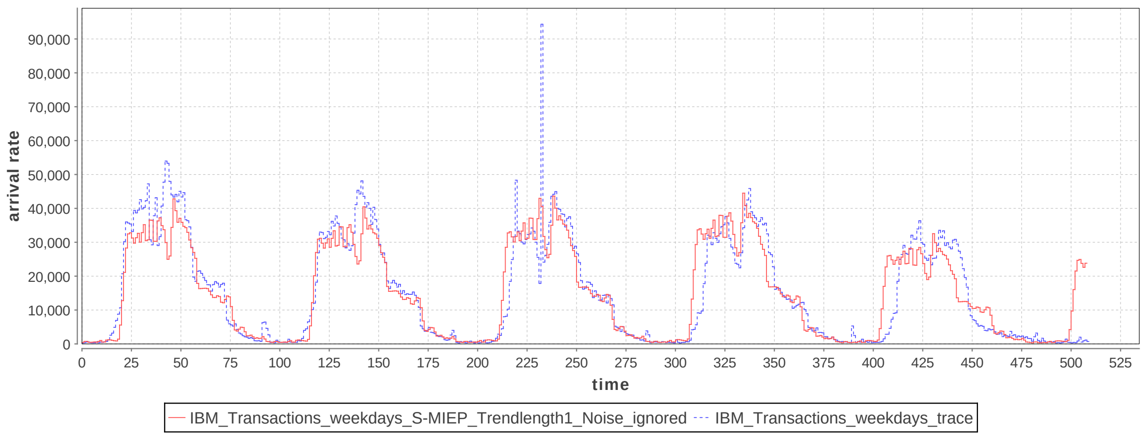


Figure 7.20.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and ignoring noise (red).

With weekends not being part of the trace, extraction results improve significantly (see Table 7.10 and Figure 7.20). The IBM trace is still a weak point of this work's extraction processes, it does however fare significantly better than with weekends included. The median accuracy of the extracted model instance without any noise reduction reaches acceptable bounds again.

Note that the DTW based curve difference is significantly better than the median and mean differences would suggest. The main reason for this phenomenon is also found in the main source of arrival rate deviation between work day trace and the extracted model instance. The extracted model instances are offset by time during the later days. Simply speaking: People seem to perform their transactions starting later on Fridays than they do on Mondays. The extracted model instance does not take this into account. The DTW distance however, detects this time-wise deviation. As explained in Section 7.1.2.2, the DTW based curve difference penalizes time-wise offset of otherwise similar arrival rates less than the simple mean and median arrival rate differences do. This leads to the DTW based curve difference indicating a significantly better match between model instance and trace than the mean and median differences.

7.1.12. Comparison with BFAST

Table 7.11 shows the comparison of the extraction accuracy of S-MIEP, with the optimal configuration for each trace, against the extraction accuracy of the BFAST[VHNC10] time-series decomposition.

To enable a fair comparison, I configured BFAST to extract one seasonal pattern and not more than 1 trend per day. In contrast to DLIM instances, where the seasonal pattern is represented by piece-wise interpolating functions, in a BFAST output, the seasonal pattern is represented as a less compact discrete function.

I also measured the run-time of both the S-MIEP extraction within LIMBO and BFAST decomposition within R. The extraction was performed on a PC with an Intel Core i7 4770 processor with 3.4 GHz and 16 GB DDR3 RAM running Windows 8.1.

LIMBO, version 14.03.0314, ran on an Eclipse Kepler Service Release 1 instance (Build ID: 20130919-0803), using the Eclipse Modeling Tools, version 2.0.2.20140224-0000, deployed on JRE Version 7 Update 51. For BFAST, I used version 1.4.4, it was run using R [Den12], version 3.0.2.

Both BFAST and S-MIEP use one processor core only, as they run on a single thread.

Each run-time measurement was executed 10 times. The measured times in table 7.11 are averages.

Trace	**S-MIEP relative median error (%)**	**BFAST relative median error (%)**	**S-MIEP run-time (ms)**	**BFAST run-time (ms)**
ClarkNet-HTTP	12.409	12.243	4.2	76276
NASA-HTTP	18.812	-	25.2	-
Saskatchewan-HTTP	26.492	-	118.8	-
WorldCup98	12.979	-	11.8	-
de.wikipedia.org	8.538	11.223	3.9	23518
fr.wikipedia.org	7.6	8.511	3.5	23630
ru.wikipedia.org	9.912	5.809	5.8	23803
wikipedia.org	4.855	2.302	3.2	21517
IBM Transactions	74.368	-	16.7	-

Table 7.11.: Accuracy and run-time comparison between S-MIEP and BFAST.

S-MIEP performs well compared to BFAST. BFAST has problems extracting large traces and did not terminate when trying to extract these traces. BFAST only terminates in traces with fewer data points. The ClarkNet-HTTP trace only extends over 2 weeks, and the Wikipedia traces contain less data points due to their lower resolution. S-MIEP performs better than BFAST for both the German and French Wikipedia traces. Here, S-MIEP's accuracy profits from its support of multiplicative trends. BFAST does, however, provide better accuracy for the English and Russian traces. It also performs slightly better than S-MIEP in the ClarkNet-HTTP extraction. In these cases, BFAST's more advanced Trend calibration mechanisms outperform S-MIEP.

S-MIEP is, however, significantly faster than BFAST. Running on the same machine, LIMBO's S-MIEP implementation performed 8354 times faster on average than BFAST's R implementation, offering the greatest speedup for the ClarkNet extraction with its higher resolution, and the smallest speedup for the Russian Wikipedia trace. The best Russian Wikipedia extraction in LIMBO entails noise reduction, which costs additional time compared to S-MIEP extractions that do not utilize noise reduction.

7.1.13. Conclusions

Evaluating DLIM and hl-DLIM accuracy has shown that the model extraction processes are capable of extracting DLIM and hl-DLIM instances with convincing accuracy from a number of different real life load intensity traces. The Wikipedia traces are the strong suit of the model extraction. The daily use patterns on Wikipedia vary very little and repeat predictably. As a result extracted *Seasonal* patterns match the trace's days well and overlaying Trends are easily calibrated.

Other traces on the other hand suffer from significant *Seasonal* pattern deviation. The IBM z-Series Transactions trace is a prime example of this. Different usage patterns on weekdays and weekends make it impossible to extract a single accurate seasonal pattern for all days. This leads to a significant decrease in accuracy. Long traces such as the seven moth long Saskatchewan-HTTP trace also suffer from *Seasonal* pattern deviation, since they leave too much time for *Seasonal* changes.

S-MIEP performs most accurately of all extraction methods. It can reach similar extraction accuracy as the BFAST time-series decomposition, while performing 8354 times as fast.

Answering research question 2.1 from Section 1.2, I can state that I have shown that it is possible to accurately extract model instances from existing traces in most cases, the major drawback being the problem of *Seasonal* pattern deviation. The assumption of a single *Seasonal* pattern for the entire trace is not always accurate. Future work will have to address this issue and define DLIM extraction processes that extract multiple *Seasonal* patterns from traces.

7.2. LIMBO Usability Evaluation

LIMBO Usability is evaluated on the basis of a questionnaire (see Appendix B) filled out by subjects from within the performance engineering community. All of the subjects are people involved in the LIMBO project in a way, in which they are considering the use of LIMBO as part of their own projects in the future.

The subjects are members of the following academic affiliations or corporations:

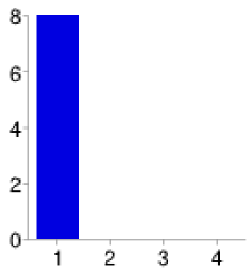
- Karlsruhe Institute of Technology (KIT)
- FZI Forschungszentrum Informatik, Karlsruhe
- Christian Albrechts University, Kiel
- SAP
- ABB

The subjects filled out the form right after completing a short 20 to 25 minute tutorial on LIMBO (see Appendix A).

The questionnaire is designed to have subjects rate the difficulty of feature use on a scale of 1 (easy) to 4 (difficult). Subjects also rate the perceived usefulness of features on a scale of 1 (very useful) to 4 (not useful). Using an even numbered scale forces subjects to choose between a positive or negative answer to each question, they cannot simply choose the middle option. In the end the questionnaire provides a few free form answer fields for envisioned use cases and needed future features.

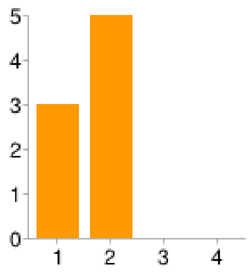
Figures 7.21 and 7.22 show that subjects are generally satisfied with the usability of LIMBO features. All features are rated on mean ease of 1 to 2, which is on the positive end of

1. LIMBO Installation is ...



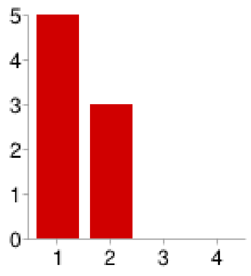
1	**8**	100%
2	**0**	0%
3	**0**	0%
4	**0**	0%

2. Creating a custom DLIM instance with the model creation wizard is ...



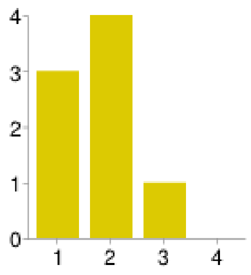
1	**3**	38%
2	**5**	63%
3	**0**	0%
4	**0**	0%

3. Extracting new model wizard parameters from an existing arrival rate trace is ...



1	**5**	63%
2	**3**	38%
3	**0**	0%
4	**0**	0%

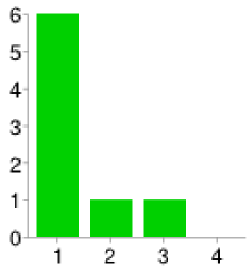
4. Editing a DLIM instance in the DLIM editor is ...



1	**3**	38%
2	**4**	50%
3	**1**	13%
4	**0**	0%

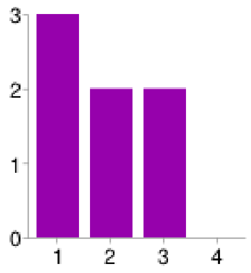
Figure 7.21.: Responses to the first 4 feature-ease-of-use questions on a scale of 1 (easy) to 4 (difficult).

5. Extracting a DLIM Sequence from an existing arrival rate trace using the Simple Extraction Process is ...



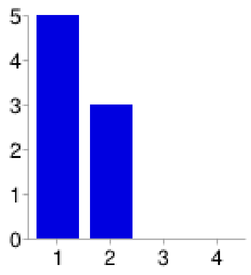
1	**6**	75%
2	**1**	13%
3	**1**	13%
4	**0**	0%

6. Extracting a DLIM Sequence from an existing arrival rate trace using the Periodic Extraction Process is ...



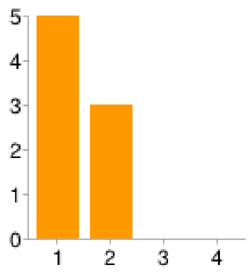
1	**3**	43%
2	**2**	29%
3	**2**	29%
4	**0**	0%

7. Creating an arrival rate time-series from a DLIM instance is ...



1	**5**	63%
2	**3**	38%
3	**0**	0%
4	**0**	0%

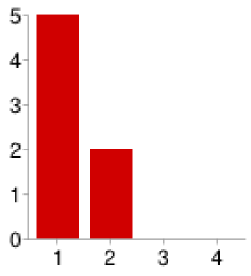
8. Creating a request time-stamp-series from a DLIM instance is ...



1	**5**	63%
2	**3**	38%
3	**0**	0%
4	**0**	0%

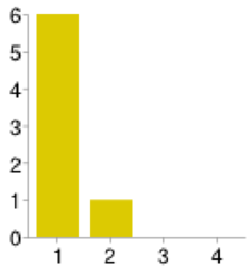
Figure 7.22.: Responses to the second 4 feature-ease-of-use questions on a scale of 1 (easy) to 4 (difficult).

9. Ability to create custom pre-populated model instances using the model creation wizard is ...



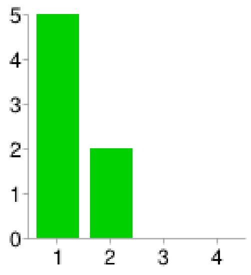
1	**5**	71%
2	**2**	29%
3	**0**	0%
4	**0**	0%

10. Ability to extract model creation wizard parameters from an existing trace is ...



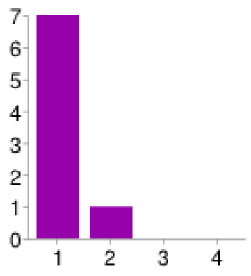
1	**6**	86%
2	**1**	14%
3	**0**	0%
4	**0**	0%

11. Ability to extract a DLIM Sequence from an existing trace is ...



1	**5**	71%
2	**2**	29%
3	**0**	0%
4	**0**	0%

12. Ability to generate time-series based on a DLIM instance is ...



1	**7**	88%
2	**1**	13%
3	**0**	0%
4	**0**	0%

Figure 7.23.: Responses to the questions on perceived feature usefulness on a scale of 1 (very useful) to 4 (not useful).

the spectrum. The only two features that received any negative feedback at all in terms of ease of use are the DLIM Editor and the extraction processes.

The relative difficulty of DLIM Editor use can be attributed to the overall difficulty of EMF Editor use, especially for people who are not used to EMF. Providing a custom editor using GMF or Graphiti might be worth considering for the future. This might help to make DLIM modeling easier for people without an EMF background.

The relative difficulty of using automated model extraction in the form of S-MIEP and P-MIEP within LIMBO on the other hand can be directly attributed to the complexity of those model instance extraction methods. The model instance extraction GUI dialogs might still be improved however, to ease the user into S-MIEP and P-MIEP use.

Figure 7.23 shows that none of the LIMBO testers disputes the usefulness of any of the major LIMBO features. The feature of time-series generation, especially, is considered as being very useful by the testers. This is not surprising as it is at the core of a majority of LIMBO use-cases. The free-form answers also emphasize this point. A majority of testers is interested in using LIMBO for benchmarking purposes. Time-series generation is a critical part for that application.

Overall LIMBO's usability can be characterized as good. Testers were generally satisfied with the ease of use of all features. The most positive outlook however, is provided by the evaluation of perceived feature usefulness. The questionnaire responses show that all of LIMBO's features are considered useful by the testers.

7.3. Load Intensity Forecasting Evaluation

This load intensity forecasting evaluation is designed to indicate whether future work on DLIM for the purpose of load intensity forecasting is warranted. The purpose of this evaluation is not a full evaluation of currently existing DLIM forecasting mechanisms, since they are out of scope for this work.

For this evaluation, I extract the first week of the wikipedia.org trace (see Section 7.1.10) into a DLIM instance using P-MIEP. As with previous periodic extractions, I set the *Trend* list for the weekly repeating *Trend* to encompass *Trend* segments of the length 3 and 4. Bi-weekly and monthly repeating *Trends* are omitted for this evaluation, since the original trace is not long enough.

Extraction Parameters	**relative median error (%)**	**relative mean error (%)**	**DTW curve difference**
P-MIEP, noise extracted	6.583	9.577	0.033744
P-MIEP, noise reduced	**6.332**	9.507	0.034726
P-MIEP, noise ignored	8.082	11.669	0.041693

Table 7.12.: wikipedia.org week 1 model extraction accuracy.

This extracted first week (extraction accuracy, see Table 7.12) is then repeated for the entire month and compared to the complete wikipedia.org trace.

Judging by the overall accuracy in Table 7.13 and Figures 7.24 and 7.25, the forecast accuracy is very good for this trace. Noise elimination seems to be useful since it always helps to mitigate the effects of *Seasonal* pattern deviation. Looking at the arrival rate comparison in Figure 7.24, however, shows significant room for improvement. Future work will have to attempt to predict future patterns and not simply repeat old ones, as is done here.

Extraction Parameters	relative median error (%)	relative mean error (%)	DTW curve difference
P-MIEP, noise extracted	7.655	9.755	0.032351
P-MIEP, noise reduced	**7.292**	9.571	0.031137
P-MIEP, noise ignored	8.546	11.513	0.036384

Table 7.13.: wikipedia.org forecast accuracy.

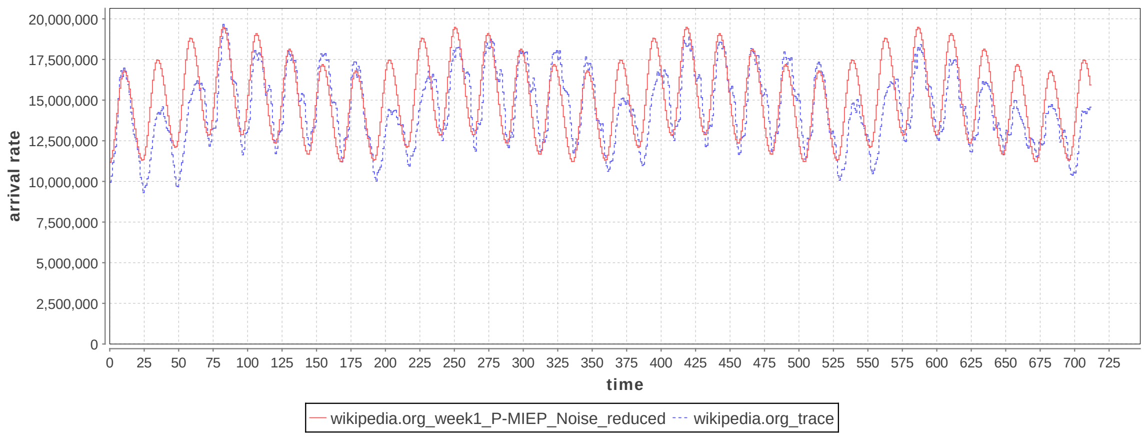


Figure 7.24.: The arrival rates as defined by the original trace (blue) and the extracted DLIM instance using S-MIEP with *Trend* length 1 and noise having been eliminated (red).

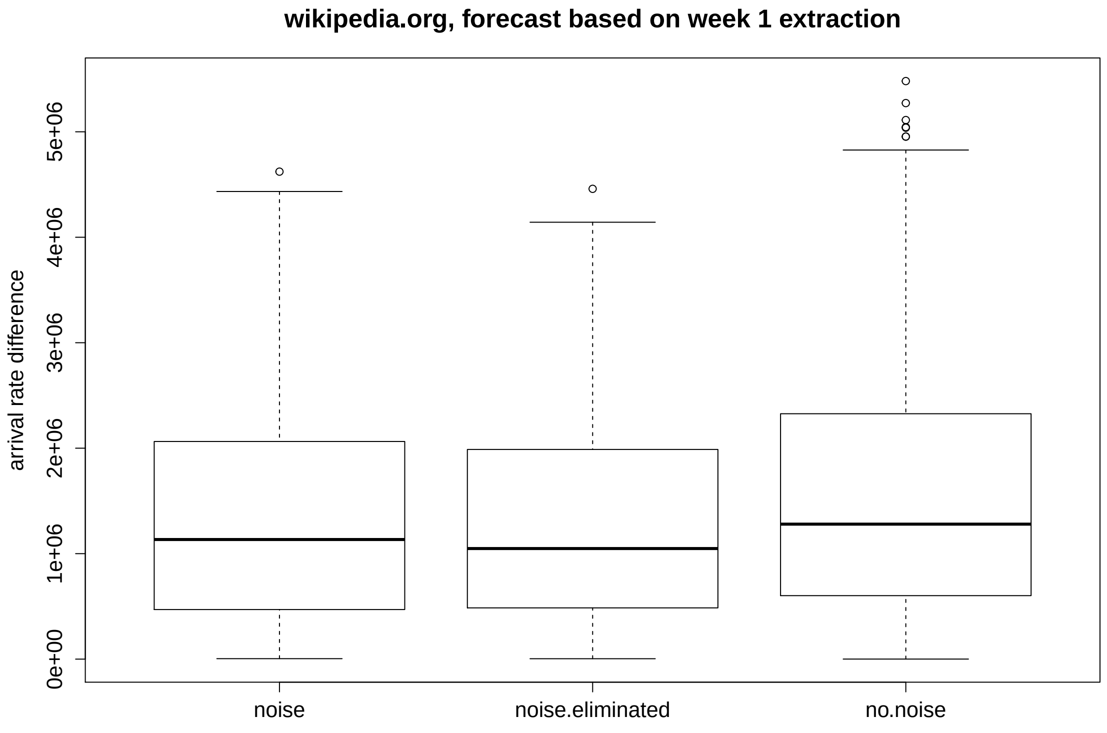


Figure 7.25.: Arrival rate forecast accuracy of the different DLIM P-MIEP configurations.

8. Future Work

While DLIM and LIMBO already provide a great wealth of features and uses, room for future works still exists. LIMBO has been designed to be extended for use with other benchmarking and workload analysis tools. For now, it is to be extended to use or be used as part of the following tools:

- Descartes Query Language (DQL) [Gor13]
- JMeter [Hal08]

Other future work can for now be split into two parts: the extension and improvement of existing features and the adaptation of DLIM and LIMBO for other uses, such as workload forecasting.

8.1. Improvement of existing features

At the center of future work improving the current DLIM and LIMBO implementations is the problem of *Seasonal* pattern deviation. Future work will have to address this issue. For now I foresee two approaches:

- Defining and extraction process for multiple *Seasonal* patterns. A process could extract multiple *Seasonal* patterns and then match them to extracted *Seasonal* periods using a best-fit approach. This could be achieved using a similar approach as in the BFAST *Seasonal* break detection, one could also attempt to include meta-data, such as calendars, in order to determine the best *Seasonal* pattern.
  - Alternatively, read time-series could be split before model extraction begins. This split would also happen on the basis of heuristics most likely based on available meta-data, such as calendar information.
- Implementing more advanced trend calibration mechanisms: The current trend calibration tries to match the maximum *Seasonal* peak with the read arrival rate at its time-stamp. This approach can be extremely inaccurate when *Seasonal* pattern deviation occurs. Implementing more advanced calibration mechanisms could circumvent this problem.

Other existing features can also be improved regardless of *Seasonal* pattern deviation.

- Implement advanced Noise Reduction techniques: The Gaussian filter currently employed for noise reduction is a simple and quick way of filtering frequencies. Far more complex and effective methods of noise reduction do exist and should be tried in the DLIM context.
- Automated *Seasonal* period extraction: The current model instance extraction processes require the user to define the *Seasonal* period based on meta information about the trace available to the user. Future work could use frequency analysis in order to automatically determine *Seasonal* periods.
- Extracting *Trends* of variable length and break detection: The current model instance extraction processes extract *Trends* of a pre-defined and fixed length. Future work could attempt to define *Trends* of differing lengths depending on current load intensity shapes within the trace. This could lead to a more dynamic and accurate DLIM instance.
- Automated *Seasonal* and *Trend* shape detection: Current model instance extraction processes require the user to define the shape of the interpolating functions for the *Seasonal* and *Trend* parts. Using heuristics, these shapes could be extracted automatically.

8.2. Extending for future Use-Cases

LIMBO is designed to be used with two major use cases in mind: Custom load intensity creation for benchmarking, and model instance extraction from existing traces. DLIM and LIMBO might however be useful for a great range of additional use-cases. Future work will have to extend DLIM and LIMBO to better fit these use-cases. Currently planned is the application of DLIM and LIMBO for load intensity forecasting. For this, LIMBO will have to be extended at the following fronts:

- Advanced heuristics for future *Trends*: The Periodic Model Instance Extraction Process assumes that all *Trends* repeat indefinitely and unchanged. More advanced heuristics could be implemented to go beyond this assumption for greater forecasting accuracy.
- Merge Simple and Periodic Model Instance Extraction Process: While the Periodic Model Instance Extraction Process can be used for simple load intensity forecasting, it can not handle a number of detectable trends. The evaluation on the basis of the WorldCup98 trace (see Section 7.1.6) showed, that it fares poorly while attempting to model a trace, which increases or decreases its load intensity over the duration of the entire trace. The Simple Model Instance Extraction Process on the other hand has no problems with this kind of trace. Future work may try to merge these two approaches in order to combine the best of both worlds.
- Extend LIMBO for run-time model calibration: LIMBO should provide utilities to calibrate an existing model instance based on new information that arrived at run-time. The performance comparison between S-MIEP and BFAST in Section 7.1.12 already shows that automated model extraction in LIMBO is fast enough in order to be used in such a context. LIMBO and DLIM could then be used for online load-forecasting.

Future work for all use-cases could entail the extension of DLIM with new *Function* implementations. The current list of DLIM *Functions* is designed to be extended and improved upon for future use.

9. Conclusion

This thesis presents the need for load intensity modeling formalisms. Load intensity models can be used for the creation of custom load intensity variations for benchmarking purposes. Benchmarks based on custom load intensity variations can be used to test specific system properties, such as elastic resource allocation and release. Load intensity models can also be used for the analysis and parametrization of existing load intensity traces.

Load intensity is defined on the basis of user or request arrival rates. When modeling variations of these arrival rates, an open workload approach has to be taken, as knowledge about work unit completion is not known by the load intensity model. Since the model only models arrival rates, it assumes that the modeled requests or user behavior present a homogenous resource use mix.

The Descartes Load Intensity Model (DLIM) provides a way of modeling arrival rate variations over time by structuring and combining mathematical functions. The evaluation (see Section 7.1) shows that it does so with great accuracy. The optimal accuracy reached during my evaluation was a median accuracy of 4.86%. This great power comes at the expense of usability, as shown in the usability evaluation in Section 7.2.

For greater usability and easier modeling this thesis also introduces the high-level Descartes Load Intensity Model (hl-DLIM). Hl-DLIM models arrival rate variations using a few parameters. While this enables better usablility (as shown in Section 7.2), it comes at the expense of DLIM's accuracy.

To enable modeling, LIMBO is introduced. LIMBO is an Eclipse-based tool for handling and instantiating load intensity models based on DLIM. LIMBO users can define variable arrival rates. LIMBO offers an accessible way of editing DLIM instances and extracting them from existing traces. It also supports additional modeling utilities, such as using hl-DLIM parameters for easy creation of new DLIM instances through a model creation wizard.

LIMBO features three model instance extraction methods. These methods can be used to extract DLIM or hl-DLIM instances from existing arrival rate traces. The model accuracy evaluation in Section 7.1 uses these methods in order to extract model instances from arrival rate traces for comparison.

The three extraction methods are:

- Simple Model Instance Extraction Process (S-MIEP): An accurate extraction process, which extracts DLIM instances from existing arrival rate traces with a median error of 19.9%. Comparison with BFAST[VHNC10] also shows, that S-MIEP provides excellent performance, with all extractions completing in less than 0.2 seconds and providing an average speedup of 8354 compared to BFAST decomposition.

- Periodic Model Instance Extraction Process (P-MIEP): A less accurate extraction process to extract DLIM instances from existing arrival rate traces. Other than S-MIEP, P-MIEP instances are intended to be repeated for load intensity forecasting. A preliminary load intensity forecasting evaluation (see Section 7.3) shows promising results, as it is able to forecast a month of requests with a median accuracy of 7.3%.
- high-level Model Instance Extraction Process (hl-MIEP): Extracts hl-DLIM instances from existing arrival rate traces. Due to hl-DLIM's limitations, this is less accurate than S-MIEP. The limited accuracy can however be improved by applying noise reduction during the extraction process.

By introducing DLIM, hl-DLIM, the extraction methods, and LIMBO, the products of this thesis provide a powerful and flexible way of modeling and analyzing various load intensity variation profiles. This can be done for a multitude of purposes, such as custom request time-stamp generation for benchmarking or the re-parametrization of arrival rate traces. LIMBO is already being used as part of other works such as Andreas Weber's on-going master's thesis on resource elasticity in cloud environments [WHGK14], an on-going master's thesis of Syed-Wahaj Ali, conducted at ABB Ladenburg, and in a case study for VMware by Simon Spinner [SKZU13].

Bibliography

[BC98] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '98/PERFORMANCE '98. New York, NY, USA: ACM, 1998, pp. 151–160. [Online]. Available: http://doi.acm.org/10.1145/277851.277897

[BFF$^+$10] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM symposium on Cloud computing.* ACM, 2010, pp. 241–252.

[Bie12] T. C. Bielefeld, "Online performance anomaly detection for large-scale software systems," 2012.

[BLY$^+$10] A. Beitch, B. Liu, T. Yung, R. Griffith, A. Fox, and D. A. Patterson, "Rain: A workload generation toolkit for cloud computing applications," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-14, Feb 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-14.html

[BS13] X. Bai and A. Shami, "Modeling self-similar traffic for network simulation," *CoRR*, vol. abs/1308.3842, 2013.

[BZ86] H. J. Blinchikoff and A. I. Zverev, *Filtering in the time and frequency domains.* Krieger Publishing Co., Inc., 1986.

[CKKR12] G. Casale, A. Kalbasi, D. Krishnamurthy, and J. Rolia, "Burn: Enabling workload burstiness in customized service benchmarks," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 778–793, 2012.

[Den12] B. Dennis, *The R Student Companion.* Boca Raton, FL: Chapman & Hall/CRC Press, 2012. [Online]. Available: http://www.crcpress.com/product/isbn/9781439875407

[Dev] C. M. Developers, "Apache commons math." [Online]. Available: http://commons.apache.org/proper/commons-math/

[Fei02] D. Feitelson, "Workload modeling for performance evaluation," in *Performance Evaluation of Complex Systems: Techniques and Tools*, ser. Lecture Notes in Computer Science, M. Calzarossa and S. Tucci, Eds. Springer Berlin Heidelberg, 2002, vol. 2459, pp. 114–141. [Online]. Available: http://dx.doi.org/10.1007/3-540-45798-4_6

[Gor13] F. Gorsler, "Online Performance Queries for Architecture-Level Performance Models," Master's thesis, Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany, July 2013.

[Hal08] E. H. Halili, *Apache JMeter: A Practical Beginner's Guide to Automated Testing and performance measurement for your websites.* Packt Publishing Ltd, 2008.

[HHKA14] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," *Concurrency and Computation: Practice and Experience*, pp. n/a–n/a, 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3224

[Hup09] K. Huppler, "The art of building a good benchmark," in *Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin Heidelberg, 2009, vol. 5895, pp. 18–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10424-4_3

[Li10] H. Li, "Realistic workload modeling and its performance impacts in large-scale escience grids," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 4, pp. 480–493, 2010.

[MAFM99] D. A. Menascé, V. A. Almeida, R. Fonseca, and M. A. Mendes, "A methodology for workload characterization of e-commerce sites," in *Proceedings of the 1st ACM conference on Electronic commerce.* ACM, 1999, pp. 119–128.

[MAR$^+$03] D. A. Menascé, V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira, Jr., "A hierarchical and multiscale approach to analyze e-business workloads," *Perform. Eval.*, vol. 54, no. 1, pp. 33–57, Sep. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0166-5316(02)00228-6

[MK12] A. Milenkoski and S. Kounev, "Towards Benchmarking Intrusion Detection Systems for Virtualized Cloud Environments," in *Proceedings of the 7th International Conference for Internet Technology and Secured Transactions (ICITST 2012).* New York, USA: IEEE, December 2012, pp. 562–563. [Online]. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6470873

[MLA10] J. McAffer, J.-M. Lemieux, and C. Aniszczyk, *Eclipse rich client platform.* Addison-Wesley Professional, 2010.

[Mül07] M. Müller, "Dynamic time warping," *Information Retrieval for Music and Motion*, pp. 69–84, 2007.

[RBG13] S. Roy, T. Begin, and P. Goncalves, "A complete framework for modelling and generating workload volatility of a vod system," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, 2013, pp. 1168–1174.

[RLGPC$^+$99] A. Reyes-Lecuona, E. González-Parada, E. Casilari, J. Casasola, and A. Diaz-Estrella, "A page-oriented www traffic model for wireless system simulations," in *Proceedings ITC*, vol. 16, 1999, pp. 1271–1280.

[SBMP08] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework.* Pearson Education, 2008.

[SC07] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, Oct. 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1367985.1367993

[Sch14] E. Schulz, "Integrating Performance Tests in a Generative Software Development Platform," Master's thesis, Christian-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, 24118 Kiel, Germany, July 2014.

[SKZU13] S. Spinner, S. Kounev, X. Zhu, and M. Uysal, "Towards Online Performance Model Extraction in Virtualized Environments," in *Proceedings of the 8th Workshop on Models @ Run.time (MRT 2013)*, N. Bencomo, R. France, S. Götz, and B. Rumpe, Eds. CEUR-WS, 2013, pp. 89–95.

[SV06] T. T. Stahl and M. Voelter, *Model-driven software development.* John Wiley & Sons Chichester, 2006.

[SWHB06] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: a cautionary tale," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 18–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267680.1267698

[Vau13] R. Vaupel, *High Availability and Scalability of Mainframe Environments using System z and z/OS as example.* KIT Scientific Publishing, Karlsruhe, 2013.

[VHNC10] J. Verbesselt, R. Hyndman, G. Newnham, and D. Culvenor, "Detecting trend and seasonal changes in satellite image time series," *Remote Sensing of Environment*, vol. 114, no. 1, pp. 106 – 115, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S003442570900265X

[vHRH08] A. van Hoorn, M. Rohr, and W. Hasselbring, "Generating probabilistic and intensity-varying workload for web-based software systems," in *Proceedings of the SPEC international workshop on Performance Evaluation: Metrics, Models and Benchmarks*, ser. SIPEW '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 124–143. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69814-2_9

[vKHK14a] J. G. von Kistowski, N. R. Herbst, and S. Kounev, "Automatic Extraction of Load Intensity Profiles using the Descartes Load Intensity Meta-Model," in *Proceedings of the 11th International Conference on Autonomic Computing (ICAC 2014).* USENIX, June 2014, submitted on March 5th, 2014.

[vKHK14b] ——, "LIMBO: A Tool For Modeling Variable Load Intensities," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014).* ACM, March 2014, accepted for Publication.

[vKHK14c] ——, "Modeling Variations in Load Intensity over Time," in *Proceedings of the 3rd International Workshop on Large-Scale Testing (LT 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014).* ACM, March 2014, accepted for Publication.

[WHGK14] A. Weber, N. R. Herbst, H. Groenda, and S. Kounev, "Towards a Resource Elasticity Benchmark for Cloud Environments," in *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014).* ACM, March 2014, accepted for Publication.

[ZF13] N. Zakay and D. G. Feitelson, "Workload resampling for performance evaluation of parallel job schedulers," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 149–160. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479893

Appendix

A. LIMBO Tutorial

LIMBO requires an up-to-date version of the Kepler Modeling Tools, available at:

`http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/keplersr1`

The Eclipse instance must be running using Java 6 or newer.

A.1. Installing LIMBO

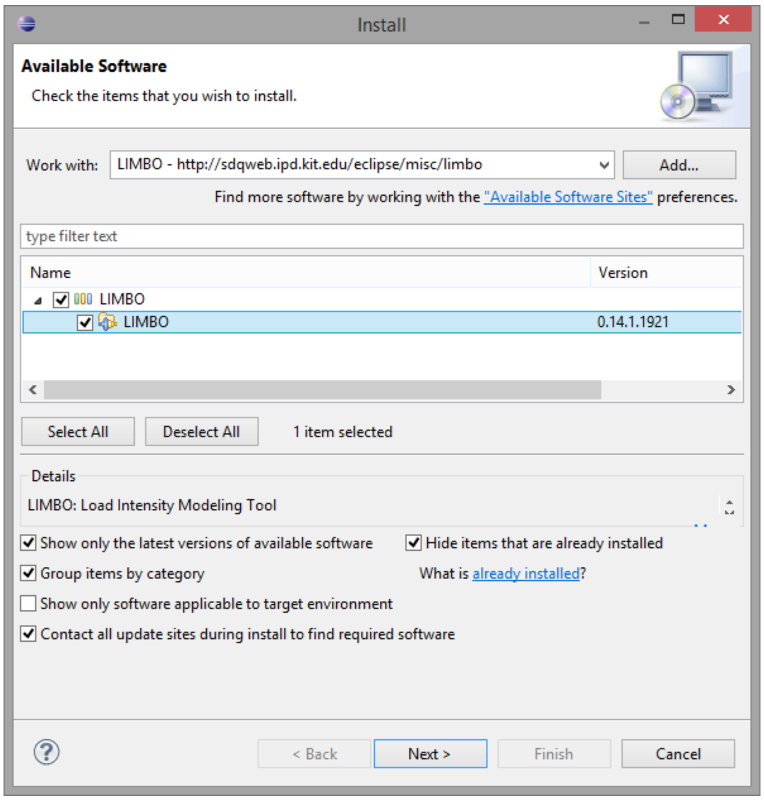
There are two ways to gain access to LIMBO:

A.1.1. Installation via Update Site

LIMBO can be downloaded from its Eclipse Update Site at:

`http://sdqweb.ipd.kit.edu/eclipse/misc/limbo`

To use this site click on **Help** → **Install new Software ...** in the Eclipse IDE, then click the **Add...** Button and enter the URL there. The site can then be selected in the **Work with:** drop-down menu and the feature should appear.



Install
Available Software
Check the items that you wish to install.
Work with: LIMBO - http://sdqweb.ipd.kit.edu/eclipse/misc/limbo
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
LIMBO
LIMBO
0.14.1.1921
Select All
Deselect All
1 item selected
Details
LIMBO: Load Intensity Modeling Tool
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

Figure A.1.: The *Install new Software ...* dialog, with LIMBO selected.

Please note: LIMBO is still in development and the update site's existence and location should remain confidential.

A.1.2. Building LIMBO from Code

The feature can be built directly from Code, available on GitHub at

`https://github.com/joakimkistowski/LIMBO`

by checking out the following plugin projects:

```
https://github.com/joakimkistowski/LIMBO/tree/master/dlim.exporter
https://github.com/joakimkistowski/LIMBO/tree/master/dlim.extractor
https://github.com/joakimkistowski/LIMBO/tree/master/dlim.generator.edit
https://github.com/joakimkistowski/LIMBO/tree/master/dlim.generator.editor
https://github.com/joakimkistowski/LIMBO/tree/master/dlim.generator
```

LIMBO can then be executed by right-clicking on any of the Eclipse-projects and choosing **Run As** → **Eclipse Application**.

A.2. Creating a new Model

A new Descartes Load Intensity Model can be created within the context of any Eclipse Project (Using separate projects for DLIM modeling is recommended).

To create a new model instance click **File** → **New** → **Other**; in the dialog choose: **Descartes Load Intensity Model** → **Descartes Load Intensity Model** and click **Next** >. Now select the project in which to place the model and enter a name.

The model creation wizard allows for easy creation of an initial Model with the use of parameters for the different parts of the model. This will be done during the course of this tutorial. Click on **Next** > to get to the next wizard page. It can however be skipped at any point by clicking the **Finish** button.

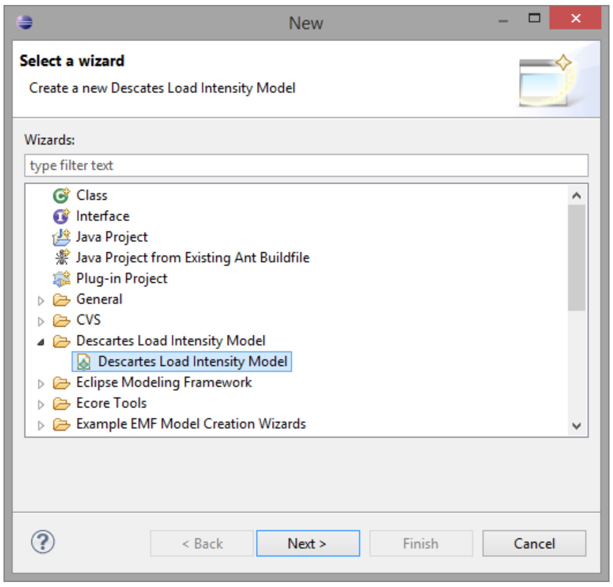
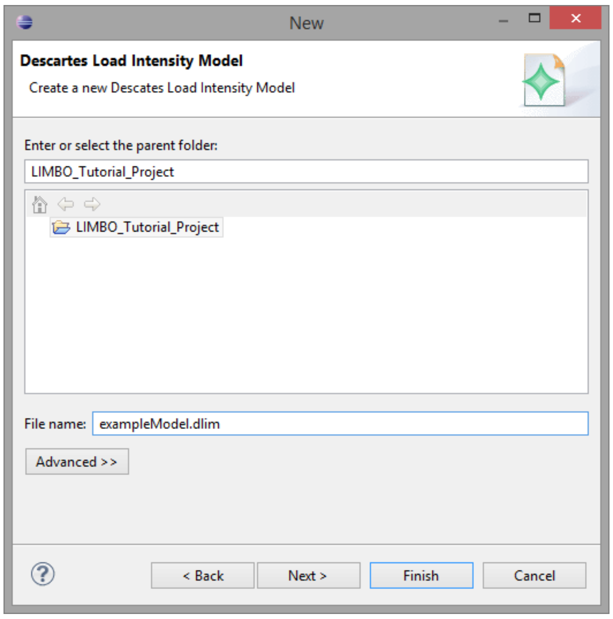

New
Select a wizard
Create a new Descates Load Intensity Model
Wizards:
type filter text
Class
Interface
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
General
CVS
Descartes Load Intensity Model
Descartes Load Intensity Model
Eclipse Modeling Framework
Ecore Tools
Example EMF Model Creation Wizards
< Back
Next >
Finish
Cancel

Figure A.2.: Choosing the DLIM creation wizard.


New

Descartes Load Intensity Model

Create a new Descates Load Intensity Model

Enter or select the parent folder:

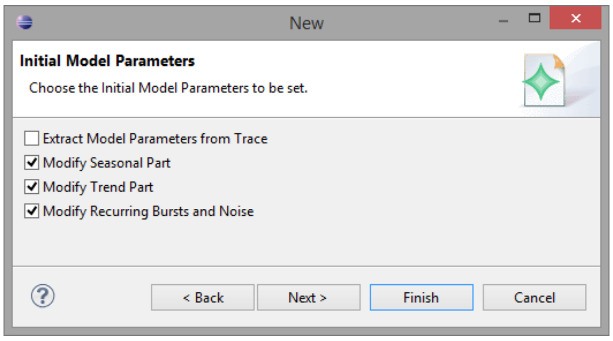
LIMBO_Tutorial_Project

LIMBO_Tutorial_Project

File name: exampleModel.dlim

Figure A.3.: Creating a new model.

The next page offers a choice about which model parts to edit in the wizard. We can extract the wizard's parameters from an arrival rate trace and modify the seasonal, trend, burst, and noise parts of the model. For now we leave this page at its default settings (**Extract Model Parameters from Trace** unchecked, everything else: checked) and click on **Next >**.


New

Initial Model Parameters

Choose the Initial Model Parameters to be set.

- ☐ Extract Model Parameters from Trace
- ☑ Modify Seasonal Part
- ☑ Modify Trend Part
- ☑ Modify Recurring Bursts and Noise

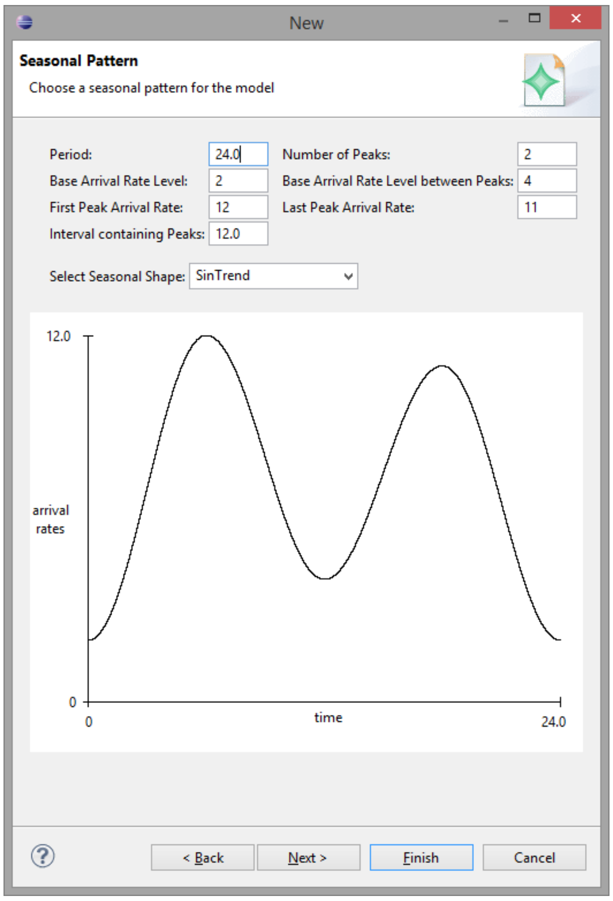
Figure A.4.: Choosing which parameters to edit.

A.2.1. Modifying the Seasonal Part

This page offers to define the model's seasonal part. The seasonal part is the repeating base function of the model. It is defined by its arrival rate peaks and base values, as well as its period (duration of a single seasonal iteration). I recommend playing around with the parameters to get a feel for them. In the end set them as follows:

Period:	24
Number of Peaks:	2
Base Arrival Rate Level:	2
Base Arrival Rate Level between Peaks:	4
First Peak Arrival Rate:	12
Last Peak Arrival Rate:	11
Interval containing Peaks:	12
Seasonal Shape:	SinTrend

Then click on **Next >**.


New

Seasonal Pattern

Choose a seasonal pattern for the model

Period:	24.0	Number of Peaks:	2
Base Arrival Rate Level:	2	Base Arrival Rate Level between Peaks:	4
First Peak Arrival Rate:	12	Last Peak Arrival Rate:	11
Interval containing Peaks:	12.0		

Select Seasonal Shape: SinTrend

Figure A.5.: The Seasonal Part.

A.2.2. Modifying the Trend Part

This Page defines the model's trend part. The trend part defines a piece-wise function, which interpolates at the maximal seasonal peaks so that these peaks reach the target arrival rate defined in the list view. Each trend segment stretches over multiple seasonal iterations and interpolates between these defined target peaks.

The trend segment length is defined by the **Number of Seasonal Periods within one Trend**. Set this to **2**.

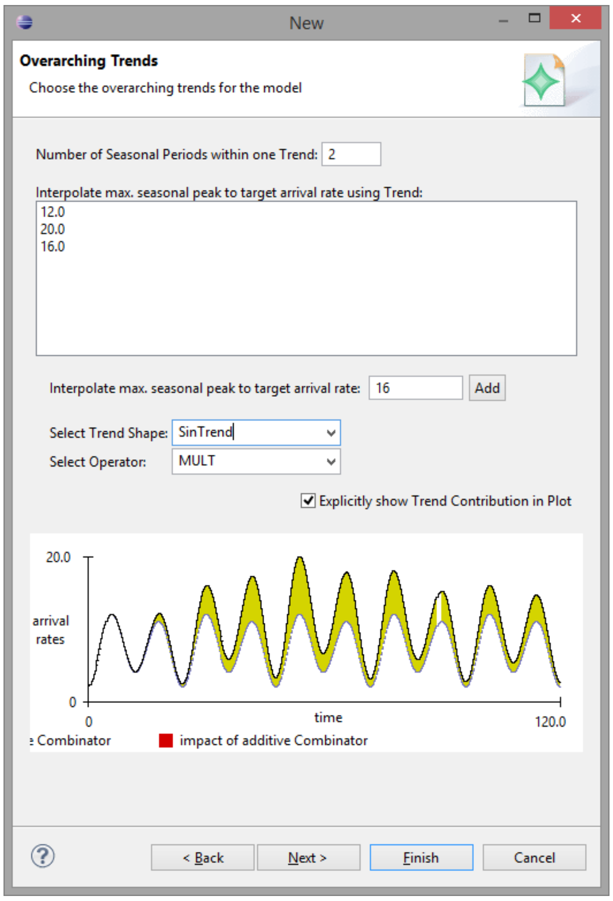
Next we must define the target arrival rates which the seasonal peaks are to reach:

In the text-field next to **Interpolate max. seasonal peak to target arrival rate:** enter **12**, then klick the **Add** button. The first trend segment will now begin at the biggest peak of the first seasonal iteration. This peak will have the arrival rate of 12.

Next enter **20** in the same text-field and click **Add** again. The first trend segment will end by interpolating the maximum arrival rate of its last seasonal iteration (remember: the segment stretches over 2 seasonal iterations) to the arrival rate of 20.

At last enter **16** and click **Add** again.

As the **Trend Shape** select **SinTrend**. This is the function the trend uses for interpolation between its defined arrival rates.


New

Overarching Trends

Choose the overarching trends for the model

Number of Seasonal Periods within one Trend: 2

Interpolate max. seasonal peak to target arrival rate using Trend:

12.0
20.0
16.0

Interpolate max. seasonal peak to target arrival rate: 16

Select Trend Shape: SinTrend

Select Operator: MULT

☑ Explicitly show Trend Contribution in Plot

Figure A.6.: The Trend Part.

Click **Next >**.

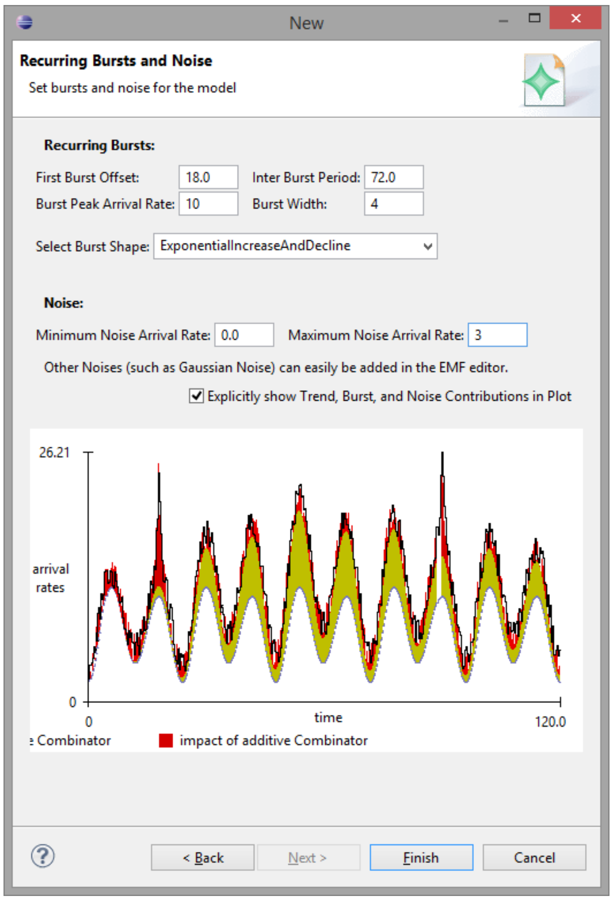
A.2.3. Modifying the Burst and Noise Parts

This page offers the definition of recurring bursts and random noise. Both are added onto the existing arrival rate output.

Define the bursts as follows:

First Burst Offset:	28
Inter Burst Period:	72
Burst Peak Arrival Rate:	10
Burst Width:	4

Additionally set the **Maximum Noise Arrival Rate** to **3**.


New
Recurring Bursts and Noise
Set bursts and noise for the model

Recurring Bursts:

First Burst Offset: 18.0 Inter Burst Period: 72.0
Burst Peak Arrival Rate: 10 Burst Width: 4

Select Burst Shape: ExponentialIncreaseAndDecline

Noise:

Minimum Noise Arrival Rate: 0.0 Maximum Noise Arrival Rate: 3

Other Noises (such as Gaussian Noise) can easily be added in the EMF editor.

[✓] Explicitly show Trend, Burst, and Noise Contributions in Plot

26.21
arrival rates
0
0
time
120.0
Combinator
🟥 impact of additive Combinator
< Back
Next >
Finish
Cancel

Figure A.7.: The Burst and Noise Parts.

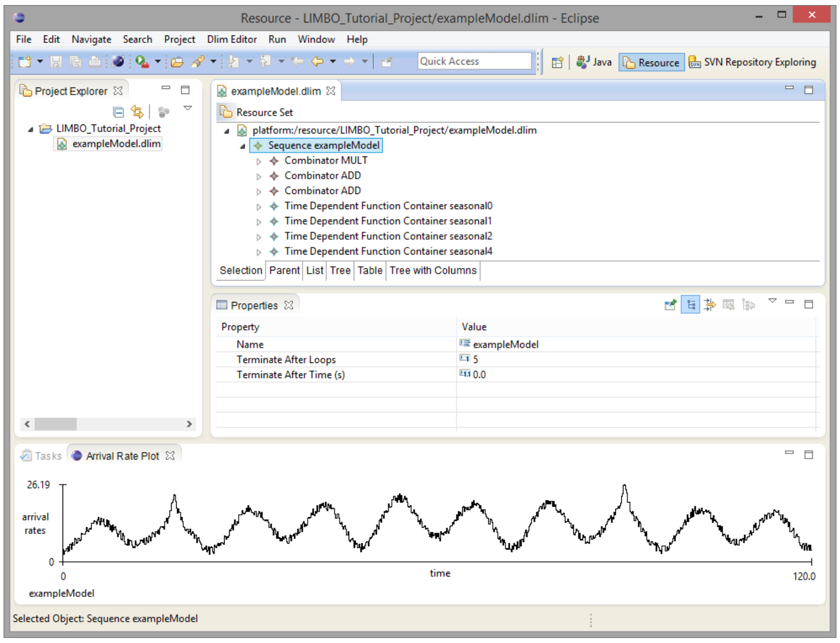
You are now done. Click **Finish** to exit the wizard.

A.3. DLIM Editor

The DLIM Editor will automatically open. The model should already be pre-populated with a root *Sequence*, a number of *TimeDependentFunctionContainers*, and a few *Combinators*.

It is recommended to turn on Live Validation for easy modeling feedback, by right-clicking inside the editor and checking **Live Validation**. Model element attributes can be changed in the Properties View, which can be opened by right-clicking on any model element (such as the root *Sequence*) and selecting **Show Properties View**. You should also open the Plot View, which visualizes the model's current arrival rate function. Do this by right-clicking anywhere in the editor and the clicking on **Show Plot View**.

Rearrange the Plot View and Properties View so that both are accessible at the same time.


Resource - LIMBO_Tutorial_Project/exampleModel.dlim - Eclipse
File Edit Navigate Search Project Dlim Editor Run Window Help
Quick Access
Java
Resource
SVN Repository Exploring
Project Explorer
exampleModel.dlim
Selection Parent List Tree Table Tree with Columns
Properties
Tasks
Arrival Rate Plot
26.19
arrival rates
0
time
120.0
exampleModel

LIMBO_Tutorial_Project
exampleModel.dlim

Resource Set
platform:/resource/LIMBO_Tutorial_Project/exampleModel.dlim
Sequence exampleModel
Combinator MULT
Combinator ADD
Combinator ADD
Time Dependent Function Container seasonal0
Time Dependent Function Container seasonal1
Time Dependent Function Container seasonal2
Time Dependent Function Container seasonal4

Property	Value
Name	exampleModel
Terminate After Loops	5
Terminate After Time (s)	0.0

Selected Object: Sequence exampleModel

Figure A.8.: The initial model.

A.3.1. Plot View

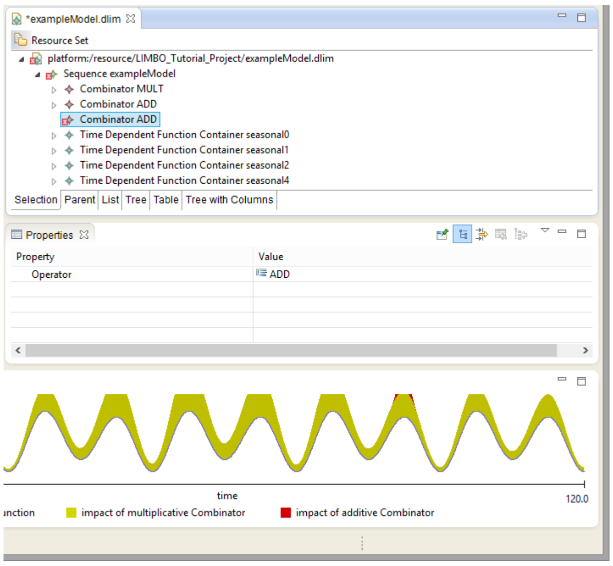
Right-clicking in the Plot View offers a few options. You can save the current plot to a file or display arrival rates from a trace for comparison.

For now, toggle the plot decomposition by clicking on **Toggle Decomposition**. The decomposition shows the impact of the different *Combinators* on the total arrival rate function.

A.3.2. Editing a DLIM instance in the Editor

All functions displayed in the DLIM editor can be deleted or edited. For this tutorial we are going to delete the uniform noise function and replace it with a normal noise distribution. We are then going to multiply a linear function onto this noise, so that it is strongest at the beginning and then fades out towards the end.

The *Uniform Noise* is contained in the third *Combinator* (The second *Combinator ADD*). Open this *Combinator*, then click on the *Uniform Noise* and delete it. The editor will now display an error, if Live Validation is enabled.


*exampleModel.dlim
Resource Set
platform:/resource/LIMBO_Tutorial_Project/exampleModel.dlim
Sequence exampleModel
Combinator MULT
Combinator ADD
Combinator ADD
Time Dependent Function Container seasonal0
Time Dependent Function Container seasonal1
Time Dependent Function Container seasonal2
Time Dependent Function Container seasonal4
Selection
Parent
List
Tree
Table
Tree with Columns
Properties
time
120.0
unction
impact of multiplicative Combinator
impact of additive Combinator

Property	Value
Operator	ADD

Figure A.9.: Our model with the Uniform Noise deleted.

Next add a *Normal Noise* to the *Combinator*. For this right-click on the *Combinator*'s then **New Child** → **Normal Noise**.

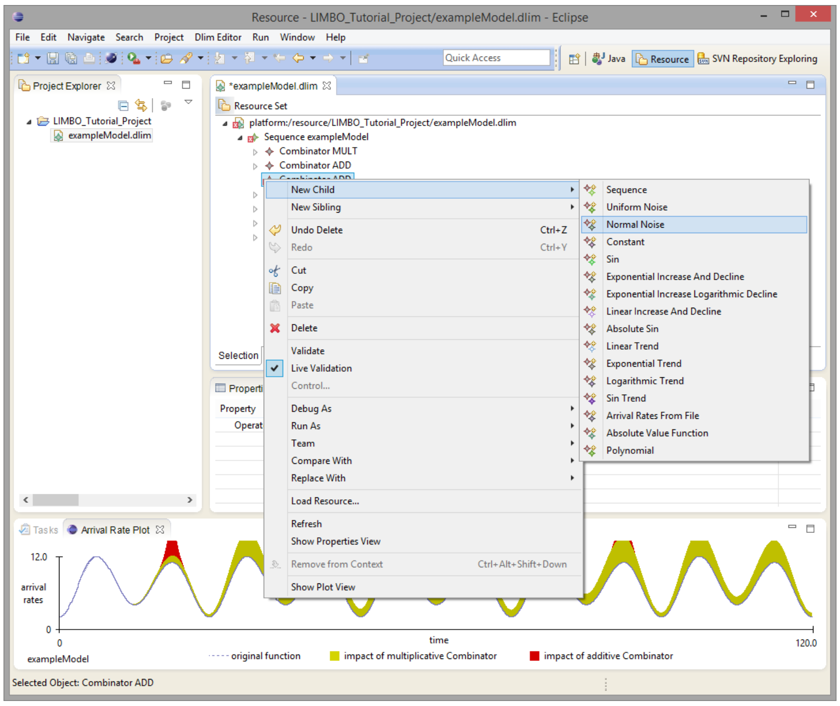
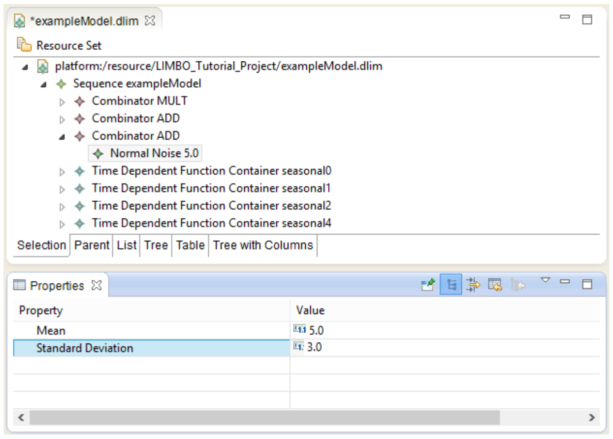


Figure A.10.: Creating a new model element.

To edit the new *Normal Noise* select it (click on it), then change its attributes in the Properties View. Set its **Mean** to **5** and its **Standard Deviation** to **3**.



*exampleModel.dlim

Resource Set

- platform:/resource/LIMBO_Tutorial_Project/exampleModel.dlim
  - Sequence exampleModel
    - Combinator MULT
    - Combinator ADD
    - Combinator ADD
      - Normal Noise 5.0
    - Time Dependent Function Container seasonal0
    - Time Dependent Function Container seasonal1
    - Time Dependent Function Container seasonal2
    - Time Dependent Function Container seasonal4

Selection | Parent | List | Tree | Table | Tree with Columns

Properties

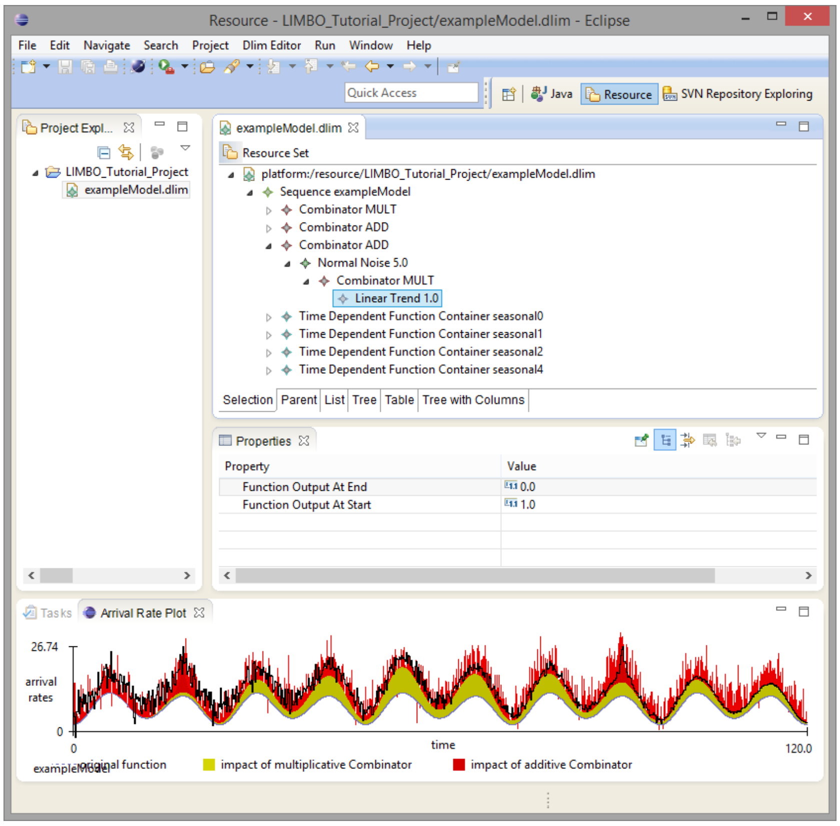
Property	Value
Mean	5.0
Standard Deviation	3.0

Figure A.11.: Editing the Normal Noise.

Next we add a *Combinator* to the *Normal Noise*. Right-click on the *Normal Noise* → **New Child** → **Combinator**. Set the new *Combinator*'s **Operator** to **MULT** in the Properties View.

We now add a *Linear Trend* to the new *Combinator*. Right-click on the *Combinator* → **New Child** → **Linear Trend**. In the Properties View set the *Linear Trend*'s **Function Output At Start** to **1** and its **Function Output At End** to *0*.

We have now successfully replaced the original *Uniform Noise* with a linearly diminishing *Normal Noise*.



Resource - LIMBO_Tutorial_Project/exampleModel.dlim - Eclipse

File Edit Navigate Search Project Dlim Editor Run Window Help

Quick Access | Java | Resource | SVN Repository Exploring

Project Expl...

- LIMBO_Tutorial_Project
  - exampleModel.dlim

exampleModel.dlim

Resource Set

- platform:/resource/LIMBO_Tutorial_Project/exampleModel.dlim
  - Sequence exampleModel
    - Combinator MULT
    - Combinator ADD
    - Combinator ADD
      - Normal Noise 5.0
        - Combinator MULT
          - Linear Trend 1.0
    - Time Dependent Function Container seasonal0
    - Time Dependent Function Container seasonal1
    - Time Dependent Function Container seasonal2
    - Time Dependent Function Container seasonal4

Selection | Parent | List | Tree | Table | Tree with Columns

Properties

Property	Value
Function Output At End	0.0
Function Output At Start	1.0

Tasks | Arrival Rate Plot

26.74
arrival rates
0
0
time
120.0
exampleModel
original function
impact of multiplicative Combinator
impact of additive Combinator

Figure A.12.: The edited Model.

A.3.3. Generating Time Stamps

Once no validation errors appear and the model has been saved (**ctrl+s**), a request time-stamp series can be generated by right-clicking the .dlim model file in the Eclipse Package Explorer and selecting **Generate Time Stamps**. A list of all currently installed time-stamp exporters appears. All default exporters, shipped with the DLIM feature write their resulting time series to their respective folders within the model's Eclipse project.

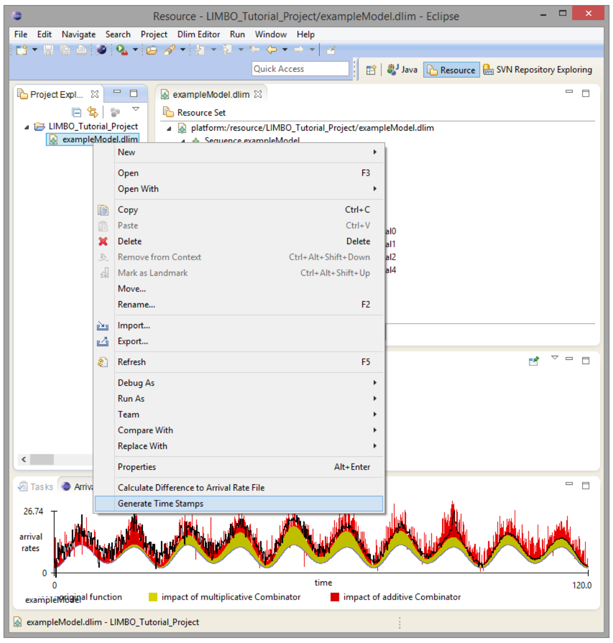

Resource - LIMBO_Tutorial_Project/exampleModel.dlim - Eclipse
File Edit Navigate Search Project Dlim Editor Run Window Help
Quick Access
Java
Resource
SVN Repository Exploring
Project Expl...
LIMBO_Tutorial_Project
exampleModel.dlim
Resource Set
platform:/resource/LIMBO_Tutorial_Project/exampleModel.dlim
Sequence exampleModel
New
Open F3
Open With
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Mark as Landmark Ctrl+Alt+Shift+Up
Move...
Rename... F2
Import...
Export...
Refresh F5
Debug As
Run As
Team
Compare With
Replace With
Properties Alt+Enter
Calculate Difference to Arrival Rate File
Generate Time Stamps
Tasks
26.74
arrival rates
0
time
120.0
original function
impact of multiplicative Combinator
impact of additive Combinator
exampleModel.dlim - LIMBO_Tutorial_Project

Figure A.13.: Generate Time-Stamps.

For this tutorial we want to create request time stamps for the use with a benchmarking framework. For this, the time-stamp generator samples the arrival rate function and then generates time-stamps according to the sampled arrival rate within each sampled interval. Select **Request Time Stamps via Equal Distance Sampling**, then click **OK**. This creates the request time-stamps with an equal distance from each other within each sampled arrival rate interval.

The resulting dialog offers a number of parameters with which to change sampling interval, the time over which the function is defined, and other parameters. The default parameters are fine for now. Click on **OK** to generate the time-stamps. A .txt file appears in the *timeStamps* folder in the .dlim file's Eclipse project.

A.3.4. Extracting a DLIM Sequence from a Trace

Next we are going to extract a *Sequence* from an existing arrival rate trace. For this we use an arrival rate trace from the German Wikipedia. Download it here:

`https://github.com/joakimkistowski/LIMBO/blob/master/DLIM_examples/trace/wikipedia_trace.txt`

The extraction process takes a DLIM *Sequence* and fills it with model elements modeling the arrival rates defined in the trace. Right-click on the model's root *Sequence* → **Extract Sequence from Arrival Rate File**.

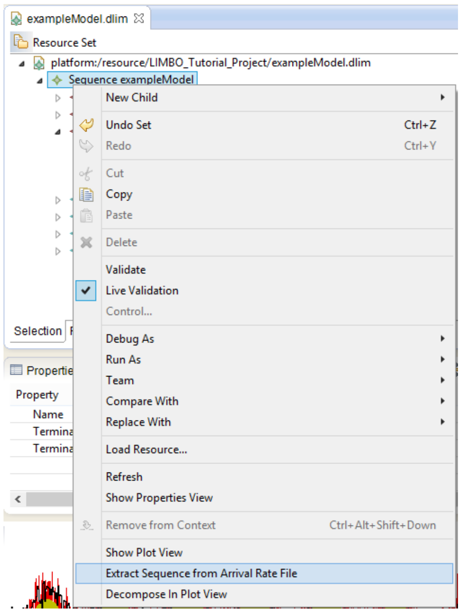
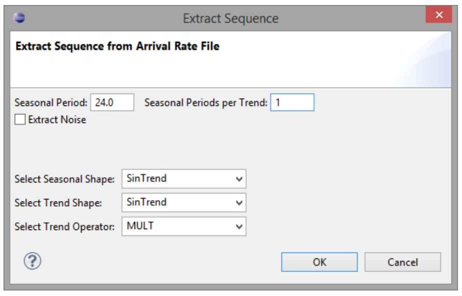


Figure A.14.: Extract Sequence from Arrival Rate Trace.

Set the downloaded trace as the **Arrival Rate File**, then select the **Simple Process Extractor** and click **OK** (The Periodic Process Extractor is explained in Section A.4.1).

In the following dialog, set the **Seasonal Period** to **24** (the trace features hourly samples) and set the **Seasonal Periods per Trend** to **1** (This setting affects trend segment length, just as it did in the model creation wizard). Click **OK**.


Extract Sequence

Extract Sequence from Arrival Rate File

Seasonal Period: 24.0 Seasonal Periods per Trend: 1

☐ Extract Noise

Select Seasonal Shape: SinTrend

Select Trend Shape: SinTrend

Select Trend Operator: MULT

Figure A.15.: Extract Sequence from Arrival Rate Trace.

A.3.5. Comparing Model and Trace

There are two ways to compare the extracted model instance to the original trace:

In the Plot View: Right-click → **Toggle Arrival Rate File Plot** → select the Wikipedia trace → **OK**. You might want to also **Toggle Decomposition** again for better visibility. The Plot View now displays the arrival rates from the trace and the arrival rates of the model for comparison.

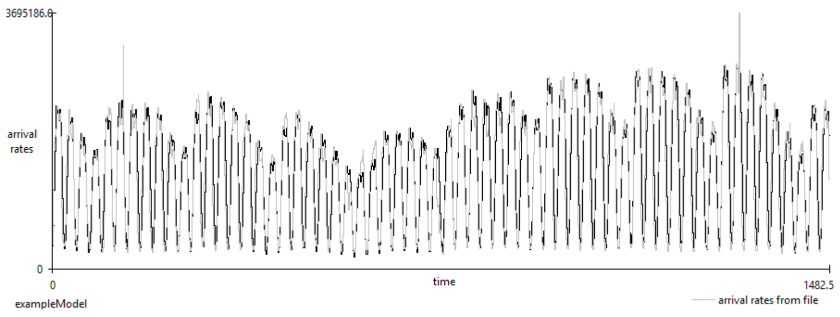


Figure A.16.: Plot View Comparison of Model and Trace.

Save the .dlim file (**ctrl+s**). In the Project Explorer Right-click on the .dlim file → **Calculate Difference to Arrival Rate File** → select the Wikipedia trace for the **Arrival Rate File** → **OK**. A dialog with a number of difference metrics appears. A list of all absolute differences is also written to the Eclipse project's *diffs* folder.

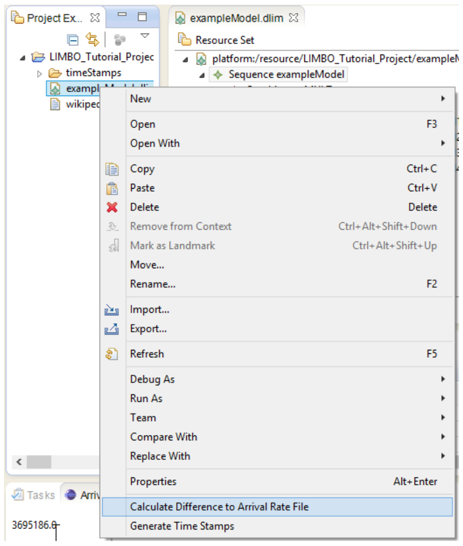


Figure A.17.: Calculate Difference between Model and Trace.

A.4. Additional Features

These additional features are not part of the tutorial, but warrant additional explanation.

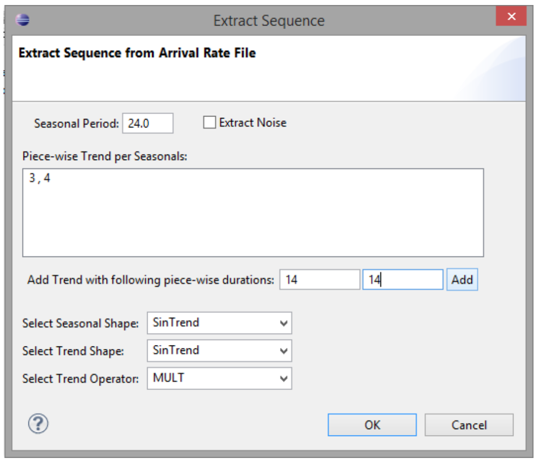
A.4.1. Periodic Process Extractor

The periodic extractor is a more complex extractor, which assumes that trends are repeating. For this the periodic extractor takes trend-segment-lists, which repeat. While the process allows for trend lists of arbitrary length, the GUI only allows for lists with 2 trend segments.

These lists can be added by filling the two text-fields below the list view in the extractor's dialog and then clicking **Add**.

Common inputs are weekly repeating trend lists with a total duration of 7 seasonal periods (days) (e.g.: 3,4) or monthly / 4-week lists with a total duration of 28 days (e.g.: 14,14).

Sequences derived using the Periodic Process Extractor are usually less accurate than *Sequences* derived using the Simple Process Extractor. They can however extend infinitely since their trends repeat.


Extract Sequence

Extract Sequence from Arrival Rate File

Seasonal Period: 24.0   ☐ Extract Noise

Piece-wise Trend per Seasonals:

3,4

Add Trend with following piece-wise durations: 14 14 Add

Select Seasonal Shape: SinTrend

Select Trend Shape: SinTrend

Select Trend Operator: MULT

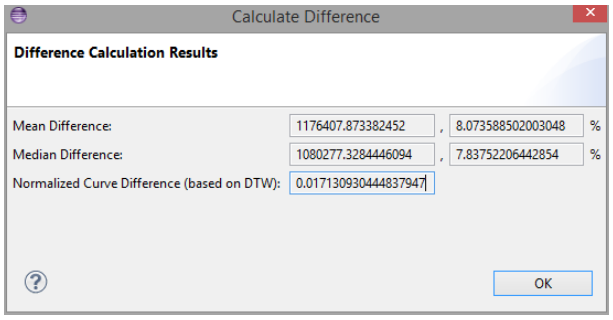
OK Cancel

Figure A.18.: Periodic Process Extraction Dialog.

A.4.2. Difference Calculator

The difference calculator calculates the difference between an arrival rate trace file and a DLIM instance. In the Project Explorer Right-click on the .dlim file → **Calculate Difference to Arrival Rate File** → select the arrival rate trace for the **Arrival Rate File** → **OK**. A list of all absolute differences is also written to the Eclipse project's *diffs* folder.

Additionally, the difference calculator displays its results in a dialog. This dialog displays the absolute and relative median and mean differences, as well as a relative curve difference based on the Dynamic Time Warping (DTW) algorithm. The DTW difference takes into account that the arrival rate variations may contain accurate arrival rates, yet be offset by time. As a result it is usually smaller than the relative mean and median differences. It is useful for comparing different model instances on the basis of the same trace.


Calculate Difference

Difference Calculation Results

Mean Difference: 1176407.873382452 , 8.073588502003048 %

Median Difference: 1080277.3284446094 , 7.83752206442854 %

Normalized Curve Difference (based on DTW): 0.017130930444837947

OK

Figure A.19.: Results of a difference calculation.

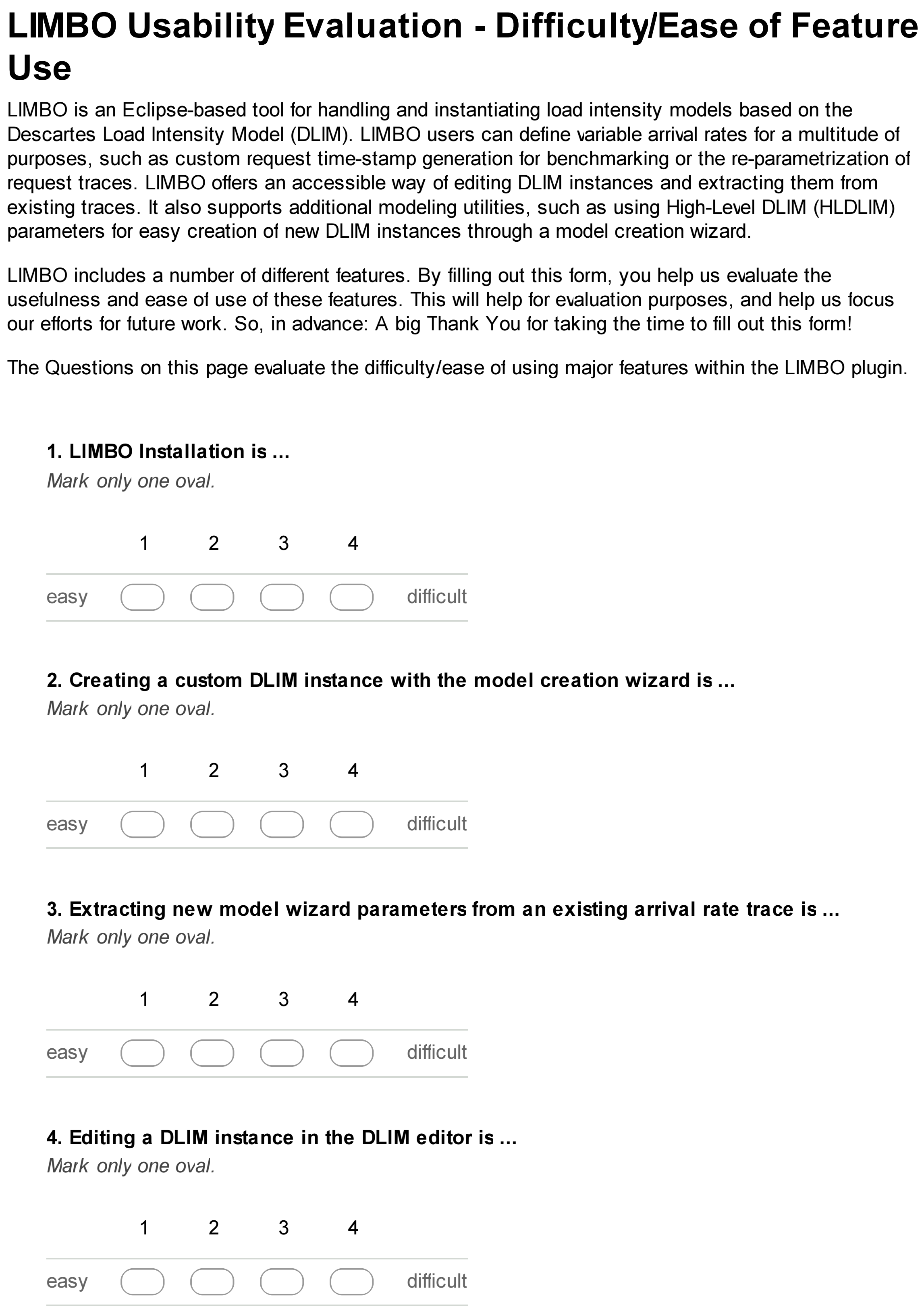
A.5. Example Models

An Eclipse project with example Models can be downloaded from GitHub at:

`https://github.com/joakimkistowski/LIMBO/tree/master/DLIM_examples`

B. LIMBO Usability Questionnaire

B.1. Questionnaire



LIMBO Usability Evaluation - Difficulty/Ease of Feature Use

LIMBO is an Eclipse-based tool for handling and instantiating load intensity models based on the Descartes Load Intensity Model (DLIM). LIMBO users can define variable arrival rates for a multitude of purposes, such as custom request time-stamp generation for benchmarking or the re-parametrization of request traces. LIMBO offers an accessible way of editing DLIM instances and extracting them from existing traces. It also supports additional modeling utilities, such as using High-Level DLIM (HLDLIM) parameters for easy creation of new DLIM instances through a model creation wizard.

LIMBO includes a number of different features. By filling out this form, you help us evaluate the usefulness and ease of use of these features. This will help for evaluation purposes, and help us focus our efforts for future work. So, in advance: A big Thank You for taking the time to fill out this form!

The Questions on this page evaluate the difficulty/ease of using major features within the LIMBO plugin.

1. LIMBO Installation is ...
Mark only one oval.

	1	2	3	4	
easy	◯	◯	◯	◯	difficult

2. Creating a custom DLIM instance with the model creation wizard is ...
Mark only one oval.

	1	2	3	4	
easy	◯	◯	◯	◯	difficult

3. Extracting new model wizard parameters from an existing arrival rate trace is ...
Mark only one oval.

	1	2	3	4	
easy	◯	◯	◯	◯	difficult

4. Editing a DLIM instance in the DLIM editor is ...
Mark only one oval.

	1	2	3	4	
easy	◯	◯	◯	◯	difficult

5. Extracting a DLIM Sequence from an existing arrival rate trace using the Simple Extraction Process is ...

Mark only one oval.

	1	2	3	4	
easy	⬭	⬭	⬭	⬭	difficult

6. Extracting a DLIM Sequence from an existing arrival rate trace using the Periodic Extraction Process is ...

Mark only one oval.

	1	2	3	4	
easy	⬭	⬭	⬭	⬭	difficult

7. Creating an arrival rate time-series from a DLIM instance is ...

Mark only one oval.

	1	2	3	4	
easy	⬭	⬭	⬭	⬭	difficult

8. Creating a request time-stamp-series from a DLIM instance is ...

Mark only one oval.

	1	2	3	4	
easy	⬭	⬭	⬭	⬭	difficult

Limbo Usability Evaluation - Feature Usefulness

The Questions on this page evaluate the usefulness of different features whithin LIMBO.

9. Ability to create custom pre-populated model instances using the model creation wizard is ...

Mark only one oval.

	1	2	3	4	
very useful	⬭	⬭	⬭	⬭	not useful

10. Ability to extract model creation wizard parameters from an existing trace is ...
Mark only one oval.

	1	2	3	4	
very useful	⬭	⬭	⬭	⬭	not useful

11. Ability to extract a DLIM Sequence from an existing trace is ...
Mark only one oval.

	1	2	3	4	
very useful	⬭	⬭	⬭	⬭	not useful

12. Ability to generate time-series based on a DLIM instance is ...
Mark only one oval.

	1	2	3	4	
very useful	⬭	⬭	⬭	⬭	not useful

Limbo Usability Evaluation - Others

Please take the time to fill out these answers.

13. **13. Which use-cases do you envision for LIMBO?**

14. **14. Which additional features would you like to see realized?**

15. **15. How would you create a variable workload from scratch without LIMBO?**
Mark only one oval.

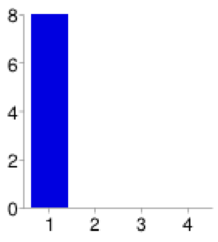
- ◯ Trace Replay
- ◯ Stochastic Model
- ◯ Other: ____________________

16. **Please Enter your Organization/Affiliation:**

____

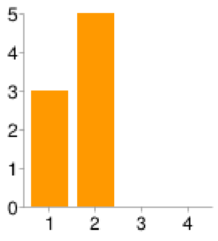
B.2. Responses

1. LIMBO Installation is ...



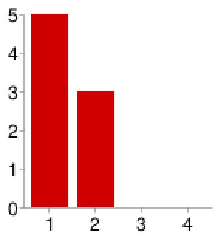
1	**8**	100%
2	**0**	0%
3	**0**	0%
4	**0**	0%

2. Creating a custom DLIM instance with the model creation wizard is ...



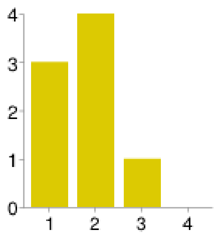
1	**3**	38%
2	**5**	63%
3	**0**	0%
4	**0**	0%

3. Extracting new model wizard parameters from an existing arrival rate trace is ...



1	**5**	63%
2	**3**	38%
3	**0**	0%
4	**0**	0%

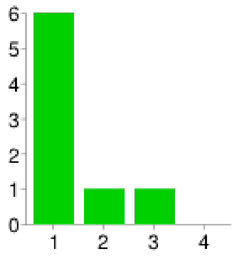
4. Editing a DLIM instance in the DLIM editor is ...



1	**3**	38%
2	**4**	50%
3	**1**	13%
4	**0**	0%

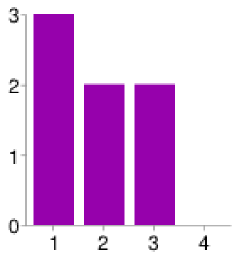
Figure B.20.: Responses to the first 4 feature-ease-of-use questions on a scale of 1 (easy) to 4 (difficult).

5. Extracting a DLIM Sequence from an existing arrival rate trace using the Simple Extraction Process is ...



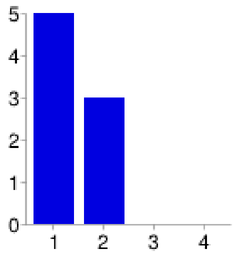
1	**6**	75%
2	**1**	13%
3	**1**	13%
4	**0**	0%

6. Extracting a DLIM Sequence from an existing arrival rate trace using the Periodic Extraction Process is ...



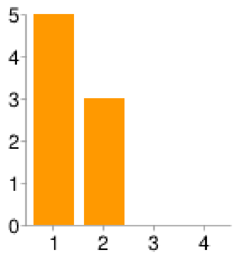
1	**3**	43%
2	**2**	29%
3	**2**	29%
4	**0**	0%

7. Creating an arrival rate time-series from a DLIM instance is ...



1	**5**	63%
2	**3**	38%
3	**0**	0%
4	**0**	0%

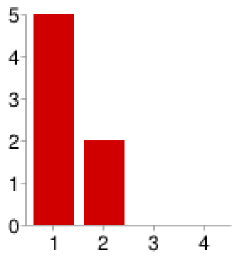
8. Creating a request time-stamp-series from a DLIM instance is ...



1	**5**	63%
2	**3**	38%
3	**0**	0%
4	**0**	0%

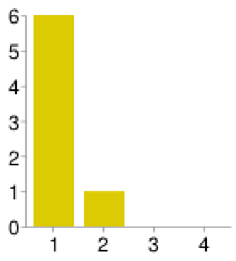
Figure B.21.: Responses to the second 4 feature-ease-of-use questions on a scale of 1 (easy) to 4 (difficult).

9. Ability to create custom pre-populated model instances using the model creation wizard is ...



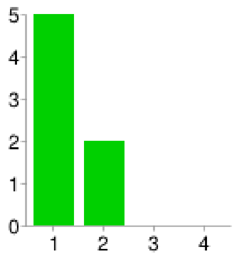
1	**5**	71%
2	**2**	29%
3	**0**	0%
4	**0**	0%

10. Ability to extract model creation wizard parameters from an existing trace is ...



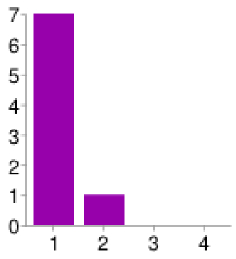
1	**6**	86%
2	**1**	14%
3	**0**	0%
4	**0**	0%

11. Ability to extract a DLIM Sequence from an existing trace is ...



1	**5**	71%
2	**2**	29%
3	**0**	0%
4	**0**	0%

12. Ability to generate time-series based on a DLIM instance is ...



1	**7**	88%
2	**1**	13%
3	**0**	0%
4	**0**	0%

Figure B.22.: Responses to the questions on perceived feature usefulness on a scale of 1 (very useful) to 4 (not useful).

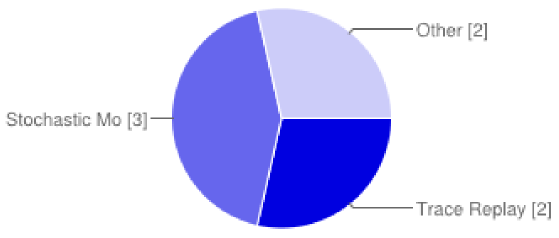
13. Which use-cases do you envision for LIMBO?

closer workload models LIMBO can produce very useful input for both workload-simulation and trace-replay benchmarks. For use with different application (benchmark) drivers that currently support only static or simple interval-based schemes for load variation. Extraction of workload intensity from a given set of user traces. Generating workloads for elasticity benchmarking! any use-case where load intensity is dramatically changing during some period of time Will be looking to use LIMBO for testing performance usage scenarios for testing automation applications in the cloud. LIMBO will certainly help with testing the elasticity and scalability of the deployed system with its ability to generate variable models.

14. Which additional features would you like to see realized?

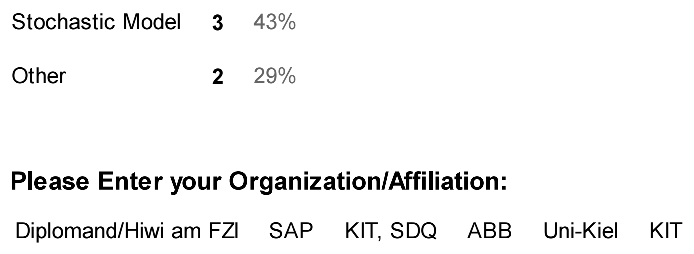
sliders to select values direct edit of the plot (click at the function and move it with the mouse into any direction) Not sure if this already works, but jMeter also allows for distributed test. Would be nice if LIMBO also supports. One way would be to create separate timestamps for each of the slave based on the model (maybe by specifying the number of slaves one wishes to have). Or a better way would be to just divide the workload using jMeter. Like I mentioned earlier, not sure if it already works with the current jMeter plugin. Zoom support for Plot view. Support for closed-workload scenarios (maybe one can already do that but I am currently unsure how to do it best. At least a discussion in the documentation would help) In creation wizard: Field "Interval containing peaks" should automatically adapt when field "period" is edited. Example: Period: 120, Number of Peaks 2 -> Interval containing peaks should automatically jump to 60 to generate a "uniform distribution" of peaks. -it were nice to have some help documentation. In LIMBO there are a lot of parameters that the user can set (especially when a new model is being created). Not all of them are self-explanatory To have various types of "work units" within one model. Also, some hints in the GUI that briefly explain which parameter does what would be helpful. Just some GUI issues, e.g., short tool-tips for (at least) function-elements in editor; plots are sometimes difficult to be viewed -> zooming functionality would be of great help.

15. How would you create a variable workload from scratch without LIMBO?



Trace Replay	**2**	29%

Figure B.23.: Responses to the free-form questions.



Stochastic Model	**3**	43%
Other	**2**	29%

Please Enter your Organization/Affiliation:

Diplomand/Hiwi am FZI　SAP　KIT, SDQ　ABB　Uni-Kiel　KIT

Figure B.24.: Responses to the free-form questions.

List of Figures

List of Tables

Glossary

BFAST Breaks For Additive Season and Trend. 2, 10, 30, 53, 61, 84, 85, 92, 94, 123, 127

Breaks For Additive Season and Trend A decomposition approach of time series into trend, season, and remainder components. See [VHNC10]. 2, 127, 129

Burst DLIM Element. Inherits from Function. Abstract parent of all interpolated burst Functions. Has a base level and a peak, which it reaches at a peak-time. 20, 21, 24, 25, 28, 45, 47, 49, 123, 127

Burst Part Hl-DLIM Part. Describes the recurring additive arrival rate bursts, wich are added onto the Seasonal Part and Trend Part. Also extracted for DLIM instances as part of S-MIEP and P-MIEP. 32, 35, 44, 50, 53, 56, 123, 127

Combinator DLIM Element. Contains a Function describing arrival rate variations and a mathematical operator. Combines its contained Function with its parent Function using its operator. 21, 24, 25, 47, 49, 51, 104–107, 123, 127

Descartes Load Intensity Model Meta-Model allowing the definition of varying load intensities through combination of piece-wise mathematical functions. 2, 15–17, 19–21, 23, 28, 30, 34, 35, 53, 61, 93, 123, 127, 129

DLIM Descartes Load Intensity Model. 2, 6, 7, 11, 13, 15–17, 19–21, 23, 26, 28, 30, 31, 34–39, 41, 44–47, 50, 51, 53, 54, 57, 59, 61, 62, 65–83, 85, 89, 90, 93, 94, 123, 124, 127, 129, 130

DTW Dynamic Time Warping. 2, 64, 65, 67, 70, 73, 74, 78, 80, 82, 83, 89, 90, 127

Dynamic Time Warping Algorithm to calculate the distance between multi-dimensional series. See [Mül07]. 2, 64, 127, 129

Function DLIM Element. Abstract parent of all mathematical functions that describe load intensity variations. May contain Combinators for combination with other functions. Is usually held by either a Combinator or a TimeDependentFunctionContainer. 21–25, 42, 49, 50, 54, 92, 123, 127

high-level Descartes Load Intensity Model Meta-Model allowing the definition of varying load intensities through a small set of seasonal, trend, burst, and noise parameters. 1, 19, 30–33, 35, 50, 53, 61, 93, 123, 127, 129

high-level Model Instance Extraction Process Extracts the set of hl-DLIM parameters from an arrival rate trace. 2, 53, 57–59, 94, 127, 130

hl-DLIM high-level Descartes Load Intensity Model. 1, 2, 6, 7, 15–17, 19, 30, 32–36, 41, 44, 50, 51, 53, 54, 57–59, 61–63, 65, 67, 69, 71, 74, 76, 78, 80, 81, 85, 93, 94, 123, 127, 129, 130

hl-MIEP high-level Model Instance Extraction Process. 2, 53, 57, 58, 62, 65, 67, 70, 73, 74, 76, 78, 80, 82, 94, 127

LIMBO Eclipse-based tooling platform for editing of hl-DLIM and DLIM instances. 2, 6, 13, 21, 25, 28, 37–39, 44, 50, 51, 56, 61, 63, 84, 85, 89, 92–94, 123, 127

Noise DLIM Element. Inherits from Function. Abstract parent of all random noise functions. 15, 20, 21, 24, 27, 42, 46, 47, 49, 123, 127

Noise Part Hl-DLIM Part. Random uniformly distributed noise. Extracted as normal distributed noise for DLIM instances as part of S-MIEP and P-MIEP. 35, 44, 50, 53, 56, 57, 127

P-MIEP Periodic Model Instance Extraction Process. 2, 53, 56, 57, 62, 65, 67, 69–71, 73, 74, 76, 78, 80, 81, 89, 90, 94, 124, 127, 129, 130

Periodic Model Instance Extraction Process Extracts a DLIM Sequence from an arrival rate trace. Uses multiple repeating trend-lists. 2, 53, 59, 94, 127, 130

Reference Clock Describes the time that is used as input for any arrival rate Function. Is defined by two DLIM Elements: ClockType and ReferenceClockObject. 21, 25, 27, 127

S-MIEP Simple Model Instance Extraction Process. 2, 53, 57, 58, 62, 65–85, 89, 90, 92–94, 123, 124, 127, 129, 130

Seasonal DLIM Element. Inherits from Function. Abstract parent of all seasonal Functions, such as sin functions. 20, 21, 24, 25, 123, 127

Seasonal Part Hl-DLIM Part. Describes the repeating base dummy-function. Also extracted for DLIM instances as part of S-MIEP and P-MIEP. 30, 31, 34, 44, 50, 53, 54, 56, 58, 65, 67, 69, 71, 74, 76, 78, 80, 81, 127, 129, 130

Sequence DLIM Element. Inherits from Function. Contains a list of TimeDependentFunctionContainers, which are executed in order. Terminates after a certain number of loops or a set time. 20–25, 28, 29, 42, 46, 47, 51, 54, 104, 108, 111, 123, 127, 130

Simple Model Instance Extraction Process Extracts a DLIM Sequence from an arrival rate trace. Uses a list of non-repeating trends. 2, 53, 55, 58, 59, 93, 123, 127, 130

TimeDependentFunctionContainer DLIM Element. Contains a Function describing arrival rate variations during its duration. Terminates after a certain duration. 21–25, 27–29, 42, 46, 47, 50, 51, 54, 104, 127

Trend DLIM Element. Inherits from Function. Abstract parent of all interpolated trend Functions. Interpolates between a functionOutputAtStart and functionOutputAtEnd. 20, 21, 24, 25, 27, 45, 47, 49, 50, 123, 127

Trend Part Hl-DLIM Part. Describes the list of interpolated functions which are multiplied or added onto the Seasonal Part. Also extracted for DLIM instances as part of S-MIEP and P-MIEP. 30, 31, 34, 35, 44, 50, 53, 54, 56, 58, 59, 65, 67, 69, 71, 74, 76, 78, 80, 81, 123, 127, 129